

OpenCore

Reference Manual (0.5.3)

[2019.11.27]

Contents

1		roduction Generic Terms	2
2	Cor	figuration	3
4		afiguration	
	2.1	Configuration Terms	3
	2.2	Configuration Processing	3
	2.3	Configuration Structure	4
3	Set	up	5
•	3.1	Directory Structure	5
	3.2	Installation and Upgrade	6
	3.3		
		Contribution	6 7
	3.4	Coding conventions	1
4	\mathbf{AC}	PI	9
	4.1	Introduction	9
	4.2	Properties	9
	4.3	Add Properties	9
	4.4		10
	4.5		10
	4.6	Quirks Properties	12
5	Boo	oter	13
	5.1	Introduction	13
	5.2		13
	5.3		13
	5.4		$^{-3}$
	0.1	quine 1 loperios	
6	Dev		17
	6.1		17
	6.2	Properties	17
	6.3	Common Properties	17
7	Ker	nol	18
•	7.1		18
	7.2		
		•	18
	7.3	•	18
	7.4	1	19
	7.5	•	20
	7.6	1	20
	7.7	Quirks Properties	21
8	Mis		24
O	8.1		24 24
	-		
	8.2		24
	8.3	•	24
	8.4	0 1	27
	8.5	v I	28
	8.6	Entry Properties	30
9	NV	$\mathbf{R}\mathbf{A}\mathbf{M}$	31
J	9.1		31
	9.2		31
	9.3		$\frac{31}{32}$
	9.4	· ·	$\frac{32}{32}$
	9.5	Other Variables	32

	PlatformInfo
	10.1 Properties
	10.2 Generic Properties
	10.3 DataHub Properties
	10.4 PlatformNVRAM Properties
	10.5 SMBIOS Properties
11	UEFI
	11.1 Introduction
	11.2 Properties
	11.3 Input Properties
	11.4 Protocols Properties
	11.5 Quirks Properties
12	Troubleshooting
	12.1 Windows support
	12.2 Debugging
	12.3 Tips and Tricks

1 Introduction

This document provides information on OpenCore user configuration file format used to setup the correct functioning of macOS operating system. It is to be read as the official clarification of expected OpenCore behaviour. All deviations, if found in published OpenCore releases, shall be considered documentation or implementation bugs, and are requested to be reported through Acidanthera Bugtracker. All other sources or translations of this document are unofficial and may contain errors.

This document is structured as a specification, and is not meant to provide a step by step algorithm for configuring end-user board support package (BSP). Any third-party articles, tools, books, etc., providing such material are prone to their authors' preferences, tastes, this document misinterpretation, and essential obsolescence. In case you still use these sources, for example, Opencore Vanilla Desktop Guide, please ensure following this document for every made decision and judging its consequences. Regardless of the sources used you are required to fully understand every dedicated OpenCore configuration option and concept prior to reporting any issues in Acidanthera Bugtracker.

1.1 Generic Terms

- plist Subset of ASCII Property List format written in XML, also know as XML plist format version 1. Uniform Type Identifier (UTI): com.apple.property-list. Plists consist of plist objects, which are combined to form a hierarchical structure. Due to plist format not being well-defined, all the definitions of this document may only be applied after plist is considered valid by running plutil -lint. External references: https://www.apple.com/DTDs/PropertyList-1.0.dtd, man plutil.
- plist type plist collections (plist array, plist dictionary, plist key) and primitives (plist string, plist data, plist date, plist boolean, plist integer, plist real).
- plist object definite realisation of plist type, which may be interpreted as value.
- plist array array-like collection, conforms to array. Consists of zero or more plist objects.
- plist dictionary map-like (associative array) collection, conforms to dict. Consists of zero or more plist keys.
- plist key contains one plist object going by the name of plist key, conforms to key. Consists of printable 7-bit ASCII characters.
- plist string printable 7-bit ASCII string, conforms to string.
- plist data base64-encoded blob, conforms to data.
- plist date ISO-8601 date, conforms to date, unsupported.
- plist boolean logical state object, which is either true (1) or false (0), conforms to true and false.
- plist integer possibly signed integer number in base 10, conforms to integer. Fits in 64-bit unsigned integer in two's complement representation, unless a smaller signed or unsigned integral type is explicitly mentioned in specific plist object description.
- plist real floating point number, conforms to real, unsupported.
- plist metadata value cast to data by the implementation. Permits passing plist string, in which case the result is represented by a null-terminated sequence of bytes (aka C string), plist integer, in which case the result is represented by 32-bit little endian sequence of bytes in two's complement representation, plist boolean, in which case the value is one byte: 01 for true and 00 for false, and plist data itself. All other types or larger integers invoke undefined behaviour.

2 Configuration

2.1 Configuration Terms

- OC config OpenCore Configuration file in plist format named config.plist. It has to provide extensible way to configure OpenCore and is structured to be separated into multiple named sections situated in the root plist dictionary. These sections are permitted to have plist array or plist dictionary types and are described in corresponding sections of this document.
- valid key plist key object of OC config described in this document or its future revisions. Besides explicitly
 described valid keys, keys starting with # symbol (e.g. #Hello) are also considered valid keys and behave as
 comments, effectively discarding their value, which is still required to be a valid plist object. All other plist
 keys are not valid, and their presence yields to undefined behaviour.
- valid value valid plist object of OC config described in this document that matches all the additional requirements in specific plist object description if any.
- invalid value valid plist object of OC config described in this document that is of other plist type, does not conform to additional requirements found in specific plist object description (e.g. value range), or missing from the corresponding collection. Invalid value is read with or without an error message as any possible value of this plist object in an undetermined manner (i.e. the values may not be same across the reboots). Whilst reading an invalid value is equivalent to reading certain defined valid value, applying incompatible value to the host system may yield to undefined behaviour.
- optional value valid value of OC config described in this document that reads in a certain defined manner provided in specific plist object description (instead of invalid value) when not present in OC config. All other cases of invalid value do still apply. Unless explicitly marked as optional value, any other value is required to be present and reads to invalid value if missing.
- fatal behaviour behaviour leading to boot termination. Implementation must stop the boot process from going any further until next host system boot. It is allowed but not required to perform cold reboot or show any warning message.
- undefined behaviour behaviour not prescribed by this document. Implementation is allowed to take any
 measures including but not limited to fatal behaviour, assuming any states or values, or ignoring, unless these
 measures negatively affect system security in general.

2.2 Configuration Processing

OC config is guaranteed to be processed at least once if it was found. Depending on OpenCore bootstrapping mechanism multiple OC config files may lead to reading any of them. No OC Config may be present on disk, in which case all the values read follow the rules of invalid value and optional value.

OC config has size, nesting, and key amount limitations. OC config size does not exceed 16 MBs. OC config has no more than 8 nesting levels. OC config has up to 16384 XML nodes (i.e. one plist dictionary item is counted as a pair of nodes) within each plist object.

Reading malformed OC config file leads to undefined behaviour. Examples of malformed OC config cover at least the following cases:

- files non-conformant to plist DTD
- files with unsupported or non-conformant plist objects found in this document
- files violating size, nesting, and key amount limitations

It is recommended but not required to abort loading malformed OC config and continue as if no OC config was present. For forward compatibility it is recommended but not required for the implementation to warn about the use of invalid values. Recommended practice of interpreting invalid values is to conform to the following convention where applicable:

Type	Value
plist string	Empty string (<string></string>)
plist data	Empty data (<data></data>)

Type	Value
plist integer plist boolean plist tristate	0 (<integer>0</integer>) False (<false></false>) False (<false></false>)

2.3 Configuration Structure

OC config is separated into following sections, which are described in separate sections of this document. By default it is tried to not enable anything and optionally provide kill switches with Enable property for plist dict entries. In general the configuration is written idiomatically to group similar actions in subsections:

- Add provides support for data addition.
- Block provides support for data removal or ignorance.
- Patch provides support for data modification.
- Quirks provides support for specific hacks.

Root configuration entries consist of the following:

- ACPI
- Booter
- DeviceProperties
- Kernel
- Misc
- NVRAM
- PlatformInfo
- UEFT

Note: Currently most properties try to have defined values even if not specified in the configuration for safety reasons. This behaviour should not be relied upon, and all fields must be properly specified in the configuration.

3 Setup

3.1 Directory Structure



Figure 1. Directory Structure

When directory boot is used the directory structure used should follow the description on Directory Structure figure. Available entries include:

- BOOTx64.efi
 - Initial booter, which loads OpenCore.efi unless it was already started as a driver.
- ACPI
 - Directory used for storing supplemental ACPI information for ACPI section.
- Drivers
 - Directory used for storing supplemental UEFI drivers for UEFI section.
- Kexts
 - Directory used for storing supplemental kernel information for Kernel section.
- Tools
 - Directory used for storing supplemental tools.
- OpenCore.efi

Main booter driver responsible for operating system loading.

Hashes for all files potentially loadable by OC Config.

• config.plist

OC Config.vault.sigSignature for vault.plist.

• vault.plist

• nvram.plist OpenCore variable import file.

 opencore-YYYY-MM-DD-HHMMSS.txt OpenCore log file.

3.2 Installation and Upgrade

To install OpenCore reflect the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information in regards to external resources like ACPI tables, UEFI drivers, or kernel extensions (kexts), completeness of the matter is out of the scope of this document. Information about kernel extensions may be found in a separate Kext List document available in OpenCore repository. Vaulting information is provided in Security Properties section of this document.

OC config, just like any property lists can be edited with any stock textual editor (e.g. nano, vim), but specialised software may provide better experience. On macOS the preferred GUI application is Xcode. For a lightweight cross-platform and open-source alternative ProperTree editor can be utilised.

For BIOS booting a third-party UEFI environment provider will have to be used. DuetPkg is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system you can install DuetPkg with a dedicated tool: BootInstall.

For upgrade purposes refer to Differences.pdf document, providing the information about the changes affecting the configuration compared to the previous release, and Changelog.md document, containing the list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. Currently supported EDK II release (potentially with patches enhancing the experience) is hosted in acidanthera/audk.

The only officially supported toolchain is XCODE5. Other toolchains might work, but are neither supported, nor recommended. Contribution of clean patches is welcome. Please do follow EDK II C Codestyle.

Required external package dependencies include EfiPkg, MacInfoPkg, and OcSupportPkg.

To compile with XCODE5, besides Xcode, one should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. Example command sequence may look as follows:

```
git clone https://github.com/acidanthera/audk UDK

git clone https://github.com/acidanthera/EfiPkg

git clone https://github.com/acidanthera/MacInfoPkg

git clone https://github.com/acidanthera/OcSupportPkg

git clone https://github.com/acidanthera/OpenCorePkg

source edksetup.sh

make -C BaseTools

build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be Sublime Text with EasyClangComplete plugin. Add .clang complete file with similar content to your UDK root:

- -I/UefiPackages/MdePkg
- -I/UefiPackages/MdePkg/Include
- -I/UefiPackages/MdePkg/Include/X64
- -I/UefiPackages/EfiPkg
- -I/UefiPackages/EfiPkg/Include
- -I/UefiPackages/EfiPkg/Include/X64
- -I/UefiPackages/AppleSupportPkg/Include
- -I/UefiPackages/OpenCorePkg/Include
- -I/UefiPackages/OcSupportPkg/Include
- -I/UefiPackages/MacInfoPkg/Include
- -I/UefiPackages/UefiCpuPkg/Include
- -IInclude
- -include

/UefiPackages/MdePkg/Include/Uefi.h

- -fshort-wchar
- -Wall
- -Wextra
- -Wno-unused-parameter
- -Wno-missing-braces
- -Wno-missing-field-initializers
- -Wno-tautological-compare
- -Wno-sign-compare
- -Wno-varargs
- -Wno-unused-const-variable
- -DOC_TARGET_NOOPT=1

Listing 2: ECC Configuration

Warning: Tool developers modifying config.plist or any other OpenCore files must ensure that their tool checks for opencore-version NVRAM variable (see Debug Properties section below) and warn the user if the version listed is unsupported or prerelease. OpenCore configuration may change across the releases and the tool shall ensure that it carefully follows this document. Failure to do so may result in this tool to be considered as malware and blocked with all possible means.

3.4 Coding conventions

Just like any other project we have conventions that we follow during the development. All third-party contributors are highly recommended to read and follow the conventions listed below before submitting their patches. In general it is also recommended to firstly discuss the issue in Acidanthera Bugtracker before sending the patch to ensure no double work and to avoid your patch being rejected.

Organisation. The codebase is structured in multiple repositories which contain separate EDK II packages. AppleSupportPkg and OpenCorePkg are primary packages, and EfiPkg, OcSupportPkg, MacInfoPkg.dsc) are dependent packages.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with XCODE5 and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.
- External pull requests and tagged commits must be validated. That said, commits in master may build but may not necessarily work.
- Internal branches should be named as follows: author-name-date, e.g. vit9696-ballooning-20191026.
- Commit messages should be prefixed with the primary module (e.g. library or code module) the changes were made in. For example, OcGuardLib: Add OC_ALIGNED macro. For non-library changes Docs or Build prefixes are used

Design. The codebase is written in a subset of freestanding C11 (C17) supported by most modern toolchains used by EDK II. Applying common software development practices or requesting clarification is recommended if any particular

case is not discussed below.

- Never rely on undefined behaviour and try to avoid implementation defined behaviour unless explicitly covered below (feel free to create an issue when a relevant case is not present).
- Use OcGuardLib to ensure safe integral arithmetics avoiding overflows. Unsigned wraparound should be relied on
 with care and reduced to the necessary amount.
- Check pointers for correct alignment with OcGuardLib and do not rely on the architecture being able to dereference unaligned pointers.
- Use flexible array members instead of zero-length or one-length arrays where necessary.
- Use static assertions (STATIC_ASSERT) for type and value assumptions, and runtime assertions (ASSERT) for
 precondition and invariant sanity checking. Do not use runtime assertions to check for errors as they should never
 alter control flow and potentially be excluded.
- Assume UINT32/INT32 to be int-sized and use %u, %d, and %x to print them.
- Assume UINTN/INTN to be of unspecified size, and cast them to UINT64/INT64 for printing with %Lu, %Ld and so
 on as normal.
- Do not rely on integer promotions for numeric literals. Use explicit casts when the type is implementationdependent or suffixes when type size is known. Assume U for UINT32 and ULL for UINT64.
- Do ensure unsigned arithmetics especially in bitwise maths, shifts in particular.
- sizeof operator should take variables instead of types where possible to be error prone. Use ARRAY_SIZE to obtain array size in elements. Use L_STR_LEN and L_STR_SIZE macros from OcStringLib to obtain string literal sizes to ensure compiler optimisation.
- Do not use goto keyword. Prefer early return, break, or continue after failing to pass error checking instead of nesting conditionals.
- Use EFIAPI, force UEFI calling convention, only in protocols, external callbacks between modules, and functions with variadic arguments.
- Provide inline documentation to every added function, at least describing its inputs, outputs, precondition, postcondition, and giving a brief description.
- Do not use RETURN_STATUS. Assume EFI_STATUS to be a matching superset that is to be always used when BOOLEAN is not enough.
- Security violations should halt the system or cause a forced reboot.

Codestyle. The codebase follows EDK II codestyle with few changes and clarifications.

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for static variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. (VOID *)(UINTN) Variable.
- Use SPDX license headers as shown in acidanthera/bugtracker#483.

Debugging. The codebase incorporates EDK II debugging and few custom features to improve the experience.

- Use module prefixes, 2-5 letters followed by a colon (:), for debug messages. For OpenCorePkg use OC:, for libraries and drivers use their own unique prefixes.
- Do not use dots (.) in the end of debug messages and separate EFI_STATUS, printed by %r, with a hyphen (e.g. OCRAM: Allocation of %u bytes failed %r\n).
- Use DEBUG_CODE_BEGIN () and DEBUG_CODE_END () constructions to guard debug checks that may potentially reduce the performance of release builds and are otherwise unnecessary.
- Use DEBUG macro to print debug messages during normal functioning, and RUNTIME_DEBUG for debugging after EXIT BOOT SERVICES.
- Use DEBUG_VERBOSE debug level to leave debug messages for future debugging of the code, which are currently not necessary. By default DEBUG_VERBOSE messages are ignored even in DEBUG builds.
- Use DEBUG_INFO debug level for all non critical messages (including errors) and DEBUG_BULK_INFO for extensive messages that should not appear in NVRAM log that is heavily limited in size. These messages are ignored in RELEASE builds.
- Use DEBUG_ERROR to print critical human visible messages that may potentially halt the boot process, and DEBUG_WARN for all other human visible errors, RELEASE builds included.

ACPI

4.1 Introduction

ACPI (Advanced Configuration and Power Interface) is an open standard to discover and configure computer hardware. ACPI specification defines the standard tables (e.g. DSDT, SSDT, FACS, DMAR) and various methods (e.g. _DSM, _PRW) for implementation. Modern hardware needs little changes to maintain ACPI compatibility, yet some of those are provided as a part of OpenCore.

To compile and disassemble ACPI tables iASL compiler can be used developed by ACPICA. GUI front-end to iASL compiler can be downloaded from Acidanthera/MaciASL.

4.2 **Properties**

1. Add

Type: plist array Failsafe: Empty

Description: Load selected tables from OC/ACPI directory.

Designed to be filled with plist dict values, describing each block entry. See Add Properties section below.

Type: plist array Failsafe: Empty

Description: Remove selected tables from ACPI stack.

Designed to be filled with plist dict values, describing each block entry. See Block Properties section below.

3. Patch

Type: plist array Failsafe: Empty

Description: Perform binary patches in ACPI tables before table addition or removal.

Designed to be filled with plist dictionary values describing each patch entry. See Patch Properties section below.

4. Quirks

Type: plist dict

Description: Apply individual ACPI quirks described in Quirks Properties section below.

Add Properties 4.3

1. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. Enabled

Type: plist boolean Failsafe: false

Description: This ACPI table will not be added unless set to true.

3. Path

Type: plist string Failsafe: Empty string

Description: File paths meant to be loaded as ACPI tables. Example values include DSDT.aml, SubDir/SSDT-8.aml,

SSDT-USBX.aml, etc.

ACPI table load order follows the item order in the array. All ACPI tables load from OC/ACPI directory.

Note: All tables but tables with DSDT table identifier (determined by parsing data not by filename) insert new tables into ACPI stack. DSDT, unlike the rest, performs replacement of DSDT table.

4.4 Block Properties

1. All

Type: plist boolean Failsafe: false

Description: If set to true, all ACPI tables matching the condition will be dropped. Otherwise only first matched table.

2. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. Enabled

Type: plist boolean Failsafe: false

Description: This ACPI table will not be removed unless set to true.

4. OemTableId

Type: plist data, 8 bytes

Failsafe: All zero

Description: Match table OEM ID to be equal to this value unless all zero.

5. TableLength

Type: plist integer

Failsafe: 0

Description: Match table size to be equal to this value unless 0.

6. TableSignature

Type: plist data, 4 bytes

Failsafe: All zero

Description: Match table signature to be equal to this value unless all zero.

Note: Make sure not to specify table signature when the sequence needs to be replaced in multiple places. Especially when performing different kinds of renames.

4.5 Patch Properties

1. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. Count

Type: plist integer

Failsafe: 0

Description: Number of patch occurrences to apply. 0 applies the patch to all occurrences found.

3. Enabled

Type: plist boolean

Failsafe: false

Description: This ACPI patch will not be used unless set to true.

4. Find

Type: plist data Failsafe: Empty data

Description: Data to find. Must equal to Replace in size.

5. Limit

Type: plist integer

Failsafe: 0

Description: Maximum number of bytes to search for. Can be set to 0 to look through the whole ACPI table.

6. Mask

Type: plist data Failsafe: Empty data

Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to Replace in size otherwise.

7. OemTableId

Type: plist data, 8 bytes

Failsafe: All zero

Description: Match table OEM ID to be equal to this value unless all zero.

8. Replace

Type: plist data Failsafe: Empty data

Description: Replacement data of one or more bytes.

9. ReplaceMask

Type: plist data Failsafe: Empty data

Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Can be set to empty data to be ignored. Must equal to Replace in size otherwise.

10. Skip

Type: plist integer

Failsafe: 0

Description: Number of found occurrences to be skipped before replacement is done.

11. TableLength

Type: plist integer

Failsafe: 0

Description: Match table size to be equal to this value unless 0.

12. TableSignature

Type: plist data, 4 bytes

Failsafe: All zero

Description: Match table signature to be equal to this value unless all zero.

In the majority of the cases ACPI patches are not useful and harmful:

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. EC and ECO), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- Avoid patching _OSI to support a higher level of feature sets unless absolutely required. Commonly this enables a number of hacks on APTIO firmwares, which result in the need to add more patches. Modern firmwares generally do not need it at all, and those that do are fine with much smaller patches.
- Try to avoid hacky changes like renaming _PRW or _DSM whenever possible.

Several cases, where patching actually does make sense, include:

- Refreshing HPET (or another device) method header to avoid compatibility checks by _OSI on legacy hardware. _STA method with if ((OSFL () == Zero)) { If (HPTE) ... Return (Zero) content may be forced to always return 0xF by replacing AO 10 93 4F 53 46 4C 00 with A4 0A 0F A3 A3 A3 A3 A3.
- To provide custom method implementation with in an SSDT, for instance, to report functional key presses on a laptop, the original method can be replaced with a dummy name by patching _Q11 with XQ11.

Tianocore AcpiAml.h source file may help understanding ACPI opcodes.

4.6 **Quirks Properties**

1. FadtEnableReset

Type: plist boolean

Failsafe: false

Description: Provide reset register and flag in FADT table to enable reboot and shutdown on legacy hardware. Not recommended unless required.

2. NormalizeHeaders

Type: plist boolean

Failsafe: false

Description: Cleanup ACPI header fields to workaround macOS ACPI implementation bug causing boot crashes. Reference: Debugging AppleACPIPlatform on 10.13 by Alex James aka theracermaster. The issue is fixed in macOS Mojave (10.14).

3. RebaseRegions

Type: plist boolean

Failsafe: false

Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

ACPI tables are often generated dynamically by underlying firmware implementation. Among the positionindependent code, ACPI tables may contain physical addresses of MMIO areas used for device configuration, usually grouped in regions (e.g. OperationRegion). Changing firmware settings or hardware configuration, upgrading or patching the firmware inevitably leads to changes in dynamically generated ACPI code, which sometimes lead to the shift of the addresses in aforementioned OperationRegion constructions.

For this reason it is very dangerous to apply any kind of modifications to ACPI tables. The most reasonable approach is to make as few as possible changes to ACPI and try to not replace any tables, especially DSDT. When this is not possible, then at least attempt to ensure that custom DSDT is based on the most recent DSDT or remove writes and reads for the affected areas.

When nothing else helps this option could be tried to avoid stalls at PCI Configuration Begin phase of macOS booting by attempting to fix the ACPI addresses. It does not do magic, and only works with most common cases. Do not use unless absolutely required.

4. ResetHwSig

Type: plist boolean Failsafe: false

Description: Reset FACS table HardwareSignature value to 0.

This works around firmwares that fail to maintain hardware signature across the reboots and cause issues with waking from hibernation.

5. ResetLogoStatus

Type: plist boolean Failsafe: false

Description: Reset BGRT table Displayed status field to false.

This works around firmwares that provide BGRT table but fail to handle screen updates afterwards.

5 Booter

5.1 Introduction

This section allows to apply different kinds of UEFI modifications on Apple bootloader (boot.efi). The modifications currently provide various patches and environment alterations for different firmwares. Some of these features were originally implemented as a part of AptioMemoryFix.efi, which is no longer maintained. See Tips and Tricks section for migration steps.

If you are using this for the first time on a customised firmware, there is a list of checks to do first. Prior to starting please ensure that you have:

- Most up-to-date UEFI firmware (check your motherboard vendor website).
- Fast Boot and Hardware Fast Boot disabled in firmware settings if present.
- Above 4G Decoding or similar enabled in firmware settings if present. Note, that on some motherboards (notably ASUS WS-X299-PRO) this option causes adverse effects, and must be disabled. While no other motherboards with the same issue are known, consider this option to be first to check if you have erratic boot failures.
- DisableIoMapper quirk enabled, or VT-d disabled in firmware settings if present, or ACPI DMAR table dropped.
- No 'slide' boot argument present in NVRAM or anywhere else. It is not necessary unless you cannot boot at all or see No slide values are usable! Use custom slide! message in the log.
- CFG Lock (MSR 0xE2 write protection) disabled in firmware settings if present. Consider patching it if you have enough skills and no option is available. See VerifyMsrE2 nots for more details.
- CSM (Compatibility Support Module) disabled in firmware settings if present. You may need to flash GOP ROM on NVIDIA 6xx/AMD 2xx or older. Use GopUpdate or AMD UEFI GOP MAKER in case you are not sure how.
- EHCI/XHCI Hand-off enabled in firmware settings only if boot stalls unless USB devices are disconnected.
- VT-x, Hyper Threading, Execute Disable Bit enabled in firmware settings if present.
- While it may not be required, sometimes you have to disable Thunderbolt support, Intel SGX, and Intel Platform Trust in firmware settings present.

When debugging sleep issues you may want to (temporarily) disable Power Nap and automatic power off, which appear to sometimes cause wake to black screen or boot loop issues on older platforms. The particular issues may vary, but in general you should check ACPI tables first. Here is an example of a bug found in some Z68 motherboards. To turn Power Nap and the others off run the following commands in Terminal:

```
sudo pmset autopoweroff 0
sudo pmset powernap 0
sudo pmset standby 0
```

Note: These settings may reset at hardware change and in certain other circumstances. To view their current state use pmset -g command in Terminal.

5.2 Properties

1. MmioWhitelist

Type: plist array

Description: Designed to be filled with plist dict values, describing addresses critical for particular firmware functioning when DevirtualiseMmio quirk is in use. See MmioWhitelist Properties section below.

2. Quirks

Type: plist dict

Description: Apply individual booter quirks described in Quirks Properties section below.

5.3 MmioWhitelist Properties

1. Address

Type: plist integer

Failsafe: 0

Description: Exceptional MMIO address, which memory descriptor should be left virtualised (unchanged) by **DevirtualiseMmio**. This means that the firmware will be able to directly communicate with this memory region during operating system functioning, because the region this value is in will be assigned a virtual address.

The addresses written here must be part of the memory map, have EfiMemoryMappedIO type and EFI_MEMORY_RUNTIME attribute (highest bit) set. To find the list of the candidates the debug log can be used.

2. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. Enabled

Type: plist boolean Failsafe: false

Description: This address will be devirtualised unless set to true.

5.4 Quirks Properties

$1. \ {\tt AvoidRuntimeDefrag}$

Type: plist boolean

Failsafe: false

Description: Protect from boot.efi runtime memory defragmentation.

This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on many firmwares using SMM backing for select services like variable storage. SMM may try to access physical addresses, but they get moved by boot.efi.

Note: Most but Apple and VMware firmwares need this quirk.

2. DevirtualiseMmio

Type: plist boolean

Failsafe: false

Description: Remove runtime attribute from select MMIO regions.

This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with the target board without additional measures. In general this frees from 64 to 256 megabytes of memory (present in the debug log), and on some platforms it is the only way to boot macOS, which otherwise fails with allocation error at bootloader stage.

This option is generally useful on all firmwares except some very old ones, like Sandy Bridge. On select firmwares it may require a list of exceptional addresses that still need to get their virtual addresses for proper NVRAM and hibernation functioning. Use MmioWhitelist section to do this.

3. DisableSingleUser

Type: plist boolean

Failsafe: false

Description: Disable single user mode.

This is a security option allowing one to restrict single user mode usage by ignoring CMD+S hotkey and -s boot argument. The behaviour with this quirk enabled is supposed to match T2-based model behaviour. Read this article to understand how to use single user mode with this quirk enabled.

4. DisableVariableWrite

 $\mathbf{Type}:$ plist boolean

Failsafe: false

Description: Protect from macOS NVRAM write access.

This is a security option allowing one to restrict NVRAM access in macOS. This quirk requires OC_FIRMWARE_RUNTIME protocol implemented in FwRuntimeServices.efi.

Note: This quirk can also be used as an ugly work around to buggy UEFI runtime services implementations that fail to write variables to NVRAM and break the rest of the operating system.

5. DiscardHibernateMap

Type: plist boolean

Failsafe: false

Description: Reuse original hibernate memory map.

This option forces XNU kernel to ignore newly supplied memory map and assume that it did not change after waking from hibernation. This behaviour is required to work by Windows, which mandates to preserve runtime memory size and location after S4 wake.

Note: This may be used to workaround buggy memory maps on older hardware, and is now considered rare legacy. Do not use this unless you fully understand the consequences.

6. EnableSafeModeSlide

Type: plist boolean

Failsafe: false

Description: Patch bootloader to have KASLR enabled in safe mode.

This option is relevant to the users that have issues booting to safe mode (e.g. by holding shift or using -x boot argument). By default safe mode forces 0 slide as if the system was launched with slide=0 boot argument. This quirk tries to patch boot.efi to lift that limitation and let some other value (from 1 to 255) be used. This quirk requires ProvideCustomSlide to be enabled.

Note: The necessity of this quirk is determined by safe mode availability. If booting to safe mode fails, this option can be tried to be enabled.

7. EnableWriteUnprotector

Type: plist boolean

Failsafe: false

Description: Permit write access to UEFI runtime services code.

This option bypasses $R\hat{X}$ permissions in code pages of UEFI runtime services by removing write protection (WP) bit from CRO register during their execution. This quirk requires OC_FIRMWARE_RUNTIME protocol implemented in FwRuntimeServices.efi.

Note: The necessity of this quirk is determined by early boot crashes of the firmware.

8. ForceExitBootServices

Type: plist boolean

Failsafe: false

Description: Retry ExitBootServices with new memory map on failure.

Try to ensure that ExitBootServices call succeeds even with outdated MemoryMap key argument by obtaining current memory map and retrying ExitBootServices call.

Note: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this unless you fully understand the consequences.

9. ProtectCsmRegion

Type: plist boolean

Failsafe: false

Description: Protect CSM region areas from relocation.

Ensure that CSM memory regions are marked as ACPI NVS to prevent boot.efi or XNU from relocating or using them.

Note: The necessity of this quirk is determined by artifacts and sleep wake issues. As AvoidRuntimeDefrag resolves a similar problem, no known firmwares should need this quirk. Do not use this unless you fully understand the consequences.

10. ProvideCustomSlide

Type: plist boolean

Failsafe: false

Description: Provide custom KASLR slide on low memory.

This option performs memory map analysis of your firmware and checks whether all slides (from 1 to 255) can be used. As boot.efi generates this value randomly with rdrand or pseudo randomly rdtsc, there is a chance of boot failure when it chooses a conflicting slide. In case potential conflicts exist, this option forces macOS to use a

pseudo random value among the available ones. This also ensures that slide= argument is never passed to the operating system for security reasons.

Note: The necessity of this quirk is determined by OCABC: Only N/256 slide values are usable! message in the debug log. If the message is present, this option is to be enabled.

11. SetupVirtualMap

Type: plist boolean

Failsafe: false

Description: Setup virtual memory at SetVirtualAddresses.

Select firmwares access memory by virtual addresses after SetVirtualAddresses call, which results in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

Note: The necessity of this quirk is determined by early boot failures.

$12. \ {\tt ShrinkMemoryMap}$

Type: plist boolean

Failsafe: false

Description: Attempt to join similar memory map entries.

Select firmwares have very large memory maps, which do not fit Apple kernel, permitting up to 64 slots for runtime memory. This quirk attempts to unify contiguous slots of similar types to prevent boot failures.

Note: The necessity of this quirk is determined by early boot failures. It is rare to need this quirk on Haswell or newer. Do not use unless you fully understand the consequences.

6 DeviceProperties

6.1 Introduction

Device configuration is provided to macOS with a dedicated buffer, called EfiDevicePropertyDatabase. This buffer is a serialised map of DevicePaths to a map of property names and their values.

Property data can be debugged with gfxutil. To obtain current property data use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree -n efi -r -x | grep device-properties |
  sed 's/.*<//;s/>.*//' > /tmp/device-properties.hex &&
  gfxutil /tmp/device-properties.hex /tmp/device-properties.plist &&
  cat /tmp/device-properties.plist
```

6.2 Properties

1. Add

Type: plist dict

Description: Sets device properties from a map (plist dict) of deivce paths to a map (plist dict) of variable names and their values in plist metadata format. Device paths must be provided in canonic string format (e.g. PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)). Properties will only be set if not present and not blocked.

Note: Currently properties may only be (formerly) added by the original driver, so unless a separate driver was installed, there is no reason to block the variables.

$2. \ {\tt Block}$

Type: plist dict

Description: Removes device properties from a map (plist dict) of deivce paths to an array (plist array) of variable names in plist string format.

6.3 Common Properties

Some known properties include:

• device-id

User-specified device identifier used for I/O Kit matching. Has 4 byte data type.

• vendor-id

User-specified vendor identifier used for I/O Kit matching. Has 4 byte data type.

• AAPL,ig-platform-id

Intel GPU framebuffer identifier used for framebuffer selection on Ivy Bridge and newer. Has 4 byte data type.

• AAPL, snb-platform-id

Intel GPU framebuffer identifier used for framebuffer selection on Sandy Bridge. Has 4 byte data type.

• layout-id

Audio layout used for AppleHDA layout selection. Has 4 byte data type.

7 Kernel

7.1 Introduction

This section allows to apply different kinds of kernelspace modifications on Apple Kernel (XNU). The modifications currently provide driver (kext) injection, kernel and driver patching, and driver blocking.

7.2 Properties

1. Add

Type: plist array Failsafe: Empty

Description: Load selected kernel drivers from OC/Kexts directory.

Designed to be filled with plist dict values, describing each driver. See Add Properties section below. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers.

2. Block

Type: plist array Failsafe: Empty

Description: Remove selected kernel drivers from prelinked kernel.

Designed to be filled with plist dictionary values, describing each blocked driver. See Block Properties section below.

3. Emulate

Type: plist dict

Description: Emulate select hardware in kernelspace via parameters described in Emulate Properties section below.

4. Patch

Type: plist array Failsafe: Empty

Description: Perform binary patches in kernel and drivers prior to driver addition and removal.

Designed to be filled with plist dictionary values, describing each patch. See Patch Properties section below.

5. Quirks

Type: plist dict

Description: Apply individual kernel and driver quirks described in Quirks Properties section below.

7.3 Add Properties

1. BundlePath

Type: plist string Failsafe: Empty string

Description: Kext bundle path (e.g. Lilu.kext or MyKext.kext/Contents/PlugIns/MySubKext.kext).

2. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. Enabled

Type: plist boolean Failsafe: false

Description: This kernel driver will not be added unless set to true.

4. ExecutablePath

Type: plist string

Failsafe: Empty string

Description: Kext executable path relative to bundle (e.g. Contents/MacOS/Lilu).

5. MaxKernel

Type: plist string Failsafe: Empty string

Description: Adds kernel driver on specified macOS version or older.

Kernel version can be obtained with uname -r command, and should look like 3 numbers separated by dots, for example 18.7.0 is the kernel version for 10.14.6. Kernel version interpretation is implemented as follows:

$$ParseDarwinVersion(\kappa,\lambda,\mu) = \kappa \cdot 10000 \quad \text{Where } \kappa \in (0,99) \text{ is kernel version major} \\ + \lambda \cdot 100 \quad \text{Where } \lambda \in (0,99) \text{ is kernel version minor} \\ + \mu \quad \text{Where } \mu \in (0,99) \text{ is kernel version patch}$$

Kernel version comparison is implemented as follows:

$$\alpha = \begin{cases} ParseDarwinVersion(\texttt{MinKernel}), & \text{If MinKernel is valid} \\ 0 & Otherwise \end{cases}$$

$$\beta = \begin{cases} ParseDarwinVersion(\texttt{MaxKernel}), & \text{If MaxKernel is valid} \\ \infty & Otherwise \end{cases}$$

$$\gamma = \begin{cases} ParseDarwinVersion(FindDarwinVersion()), & \text{If valid "Darwin Kernel Version" is found} \\ 0 & Otherwise \end{cases}$$

$$f(\alpha, \beta, \gamma) = \alpha \leq \gamma \leq \beta$$

Here ParseDarwinVersion argument is assumed to be 3 integers obtained by splitting Darwin kernel version string from left to right by the . symbol. FindDarwinVersion function looks up Darwin kernel version by locating "Darwin Kernel Version $\kappa.\lambda.\mu$ " string in the kernel image.

6. MinKernel

Type: plist string Failsafe: Empty string

Description: Adds kernel driver on specified macOS version or newer.

Note: Refer to Add MaxKernel description for matching logic.

7. PlistPath

Type: plist string Failsafe: Empty string

Description: Kext Info.plist path relative to bundle (e.g. Contents/Info.plist).

7.4 Block Properties

1. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. Enabled

Type: plist boolean Failsafe: false

Description: This kernel driver will not be blocked unless set to true.

3. Identifier

Type: plist string Failsafe: Empty string

Description: Kext bundle identifier (e.g. com.apple.driver.AppleTyMCEDriver).

4. MaxKernel

Type: plist string Failsafe: Empty string

Description: Blocks kernel driver on specified macOS version or older.

Note: Refer to Add MaxKernel description for matching logic.

5. MinKernel

Type: plist string Failsafe: Empty string

Description: Blocks kernel driver on specified macOS version or newer.

Note: Refer to Add MaxKernel description for matching logic.

7.5 Emulate Properties

1. Cpuid1Data

Type: plist data, 16 bytes

Failsafe: All zero

Description: Sequence of EAX, EBX, ECX, EDX values in Little Endian order to replace CPUID (1) call in XNU kernel. Normally it is only the value of EAX that needs to be taken care of, which represents the exact CPUID. And the remainders are to be left as zeroes. For instance, A9 06 03 00 stands for CPUID 0x0306A9 (Ivy Bridge). A good example can be found at acidanthera/bugtracker#365. (See Special NOTES for Haswell+ low-end)

2. Cpuid1Mask

Type: plist data, 16 bytes

Failsafe: All zero

Description: Bit mask of active bits in Cpuid1Data. When each Cpuid1Mask bit is set to 0, the original CPU bit is used, otherwise set bits take the value of Cpuid1Data.

7.6 Patch Properties

1. Base

Type: plist string Failsafe: Empty string

Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of provided symbol name. Can be set to empty string to be ignored.

2. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. Count

Type: plist integer

Failsafe: 0

Description: Number of patch occurrences to apply. 0 applies the patch to all occurrences found.

4. Enabled

Type: plist boolean

Failsafe: false

Description: This kernel patch will not be used unless set to true.

5. Find

Type: plist data Failsafe: Empty data

Description: Data to find. Can be set to empty for immediate replacement at Base. Must equal to Replace in size otherwise.

6. Identifier

Type: plist string

Failsafe: Empty string

Description: Kext bundle identifier (e.g. com.apple.driver.AppleHDA) or kernel for kernel patch.

7. Limit

Type: plist integer

Failsafe: 0

Description: Maximum number of bytes to search for. Can be set to 0 to look through the whole kext or kernel.

8. Mask

Type: plist data Failsafe: Empty data

Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set

to zero) bits. Can be set to empty data to be ignored. Must equal to Replace in size otherwise.

9. MaxKernel

Type: plist string Failsafe: Empty string

Description: Patches data on specified macOS version or older.

Note: Refer to Add MaxKernel description for matching logic.

10. MinKernel

Type: plist string Failsafe: Empty string

Description: Patches data on specified macOS version or newer.

Note: Refer to Add MaxKernel description for matching logic.

11. Replace

Type: plist data Failsafe: Empty data

Description: Replacement data of one or more bytes.

$12. \ {\tt ReplaceMask}$

Type: plist data Failsafe: Empty data

Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set

to non-zero) bits. Can be set to empty data to be ignored. Must equal to Replace in size otherwise.

13. Skip

Type: plist integer

Failsafe: 0

Description: Number of found occurrences to be skipped before replacement is done.

7.7 Quirks Properties

1. AppleCpuPmCfgLock

Type: plist boolean

Failsafe: false

Description: Disables PKG_CST_CONFIG_CONTROL (0xE2) MSR modification in AppleIntelCPUPowerManagement.kext, commonly causing early kernel panic, when it is locked from writing.

Note: This option should avoided whenever possible. Modern firmwares provide CFG Lock setting, disabling which is much cleaner. More details about the issue can be found in VerifyMsrE2 notes.

$2. \ {\tt AppleXcpmCfgLock}$

Type: plist boolean

Failsafe: false

Description: Disables PKG_CST_CONFIG_CONTROL (0xE2) MSR modification in XNU kernel, commonly causing early kernel panic, when it is locked from writing (XCPM power management).

Note: This option should avoided whenever possible. Modern firmwares provide CFG Lock setting, disabling which is much cleaner. More details about the issue can be found in VerifyMsrE2 notes.

3. AppleXcpmExtraMsrs

Type: plist boolean

Failsafe: false

Description: Disables multiple MSR access critical for select CPUs, which have no native XCPM support.

This is normally used in conjunction with Emulate section on Haswell-E, Broadwell-E, Skylake-X, and similar CPUs. More details on the XCPM patches are outlined in acidanthera/bugtracker#365.

Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use AppleIntelCpuPowerManagement.kext for the former.

4. CustomSMBIOSGuid

Type: plist boolean

Failsafe: false

Description: Performs GUID patching for UpdateSMBIOSMode Custom mode. Usually relevant for Dell laptops.

5. DisableIoMapper

Type: plist boolean

Failsafe: false

Description: Disables IOMapper support in XNU (VT-d), which may conflict with the firmware implementation.

Note: This option is a preferred alternative to dropping DMAR ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

6. ExternalDiskIcons

Type: plist boolean

Failsafe: false

Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

Note: This option should avoided whenever possible. Modern firmwares usually have compatible AHCI controllers.

7. LapicKernelPanic

Type: plist boolean

Failsafe: false

Description: Disables kernel panic on LAPIC interrupts.

8. PanicNoKextDump

Type: plist boolean

Failsafe: false

Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details.

Affects 10.13 and above.

9. PowerTimeoutKernelPanic

Type: plist boolean

Failsafe: false

Description: Disables kernel panic on setPowerState timeout.

An additional security measure was added to macOS Catalina (10.15) causing kernel panic on power change timeout for Apple drivers. Sometimes it may cause issues on misconfigured hardware, notably digital audio, which sometimes fails to wake up. For debug kernels setpowerstate_panic=0 boot argument should be used, which is otherwise equivalent to this quirk.

10. ThirdPartyDrives

Type: plist boolean

Failsafe: false

Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

Note: This option may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called trimforce. Starting from 10.15 this utility creates EnableTRIM variable in APPLE_BOOT_VARIABLE_GUID namespace with 01 00 00 00 value.

11. XhciPortLimit

Type: plist boolean

Failsafe: false

Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

Note: This option should be avoided whenever possible. USB port limit is imposed by the amount of used bits in locationID format and there is no possible way to workaround this without heavy OS modification. The only valid solution is to limit the amount of used ports to 15 (discarding some). More details can be found on AppleLife.ru.

8 Misc

8.1 Introduction

This section contains miscellaneous configuration entries for OpenCore behaviour that does not go to any other sections

8.2 Properties

1. Boot

Type: plist dict

Description: Apply boot configuration described in Boot Properties section below.

 $2. \; {\tt BlessOverride}$

Type: plist array

Description: Add custom scanning paths through bless model.

Designed to be filled with plist string entries containing absolute UEFI paths to customised bootloaders, for example, \EFI\Microsoft\Boot\bootmgfw.efi for Microsoft bootloader. This allows unusual boot paths to be automatically discovered by the boot picker. Designwise they are equivalent to predefined blessed path, such as \System\Library\CoreServices\boot.efi, but unlike predefined bless paths they have highest priority.

3. Debug

Type: plist dict

Description: Apply debug configuration described in Debug Properties section below.

4. Entries

Type: plist array

Description: Add boot entries to boot picker.

Designed to be filled with plist dict values, describing each load entry. See Entry Properties section below.

5. Security

Type: plist dict

Description: Apply security configuration described in Security Properties section below.

6. Tools

Type: plist array

Description: Add tool entries to boot picker.

Designed to be filled with plist dict values, describing each load entry. See Entry Properties section below.

Note: Select tools, for example, UEFI Shell are very dangerous and **MUST NOT** appear in production configurations, especially in vaulted ones and protected with secure boot, as they may be used to easily bypass secure boot chain.

8.3 Boot Properties

1. ConsoleMode

Type: plist string Failsafe: Empty string

Description: Sets console output mode as specified with the WxH (e.g. 80x24) formatted string. Set to empty string not to change console mode. Set to Max to try to use largest available console mode.

Note: This field is best to be left empty on most firmwares.

2. ConsoleBehaviourOs

Type: plist string Failsafe: Empty string

Description: Set console control behaviour upon operating system load.

Console control is a legacy protocol used for switching between text and graphics screen output. Some firmwares do not provide it, yet select operating systems require its presence, which is what ConsoleControl UEFI protocol is for.

When console control is available, OpenCore can be made console control aware, and set different modes for the operating system booter (ConsoleBehaviourOs), which normally runs in graphics mode, and its own user interface (ConsoleBehaviourUi), which normally runs in text mode. Possible behaviours, set as values of these options, include:

- Empty string Do not modify console control mode.
- Text Switch to text mode.
- Graphics Switch to graphics mode.
- ForceText Switch to text mode and preserve it (requires ConsoleControl).
- ForceGraphics Switch to graphics mode and preserve it (require ConsoleControl).

Hints:

- Unless empty works, firstly try to set ConsoleBehaviourOs to Graphics and ConsoleBehaviourUi to Text.
- On APTIO IV (Haswell and earlier) it is usually enough to have ConsoleBehaviourOs set to Graphics and ConsoleBehaviourUi set to ForceText to avoid visual glitches.
- On APTIO V (Broadwell and newer) ConsoleBehaviourOs set to ForceGraphics and ConsoleBehaviourUi set to ForceText usually works best.
- On Apple firmwares ConsoleBehaviourOs set to Graphics and ConsoleBehaviourUi set to Text is supposed to work best.

Note: IgnoreTextInGraphics and SanitiseClearScreen may need to be enabled for select firmware implementations. Particularly APTIO firmwares.

3. ConsoleBehaviourUi

Type: plist string Failsafe: Empty string

Description: Set console control behaviour upon OpenCore user interface load. Refer to ConsoleBehaviourOs description for details.

4. HibernateMode

Type: plist string

Failsafe: None

Description: Hibernation detection mode. The following modes are supported:

- \bullet None — Avoid hibernation for your own good.
- Auto Use RTC and NVRAM detection.
- RTC Use RTC detection.
- NVRAM Use NVRAM detection.

$5. \; {\tt HideSelf}$

Type: plist boolean Failsafe: false

Description: Hides own boot entry from boot picker. This may potentially hide other entries, for instance, when another UEFI OS is installed on the same volume and driver boot is used.

6. PollAppleHotKeys

Type: plist boolean

Failsafe: false

Description: Enable modifier hotkey handling in boot picker.

In addition to action hotkeys, which are partially described in UsePicker section and are normally handled by Apple BDS, there exist modifier keys, which are handled by operating system bootloader, namely boot.efi. These keys allow to change operating system behaviour by providing different boot modes.

On some firmwares it may be problematic to use modifier keys due to driver incompatibilities. To workaround this problem this option allows registering select hotkeys in a more permissive manner from within boot picker. Such extensions include the support of tapping on keys in addition to holding and pressing Shift along with other keys instead of just Shift alone, which is not detectible on many PS/2 keyboards. This list of known hotkeys includes:

- CMD+C+MINUS disable board compatibility checking.
- CMD+K boot release kernel, similar to kcsuffix=release.

- CMD+S single user mode.
- CMD+S+MINUS disable KASLR slide, requires disabled SIP.
- CMD+V verbose mode.
- Shift safe mode.

7. Resolution

Type: plist string Failsafe: Empty string

Description: Sets console output screen resolution.

- Set to WxH@Bpp (e.g. 1920x1080@32) or WxH (e.g. 1920x1080) formatted string to request custom resolution from GOP if available.
- Set to empty string not to change screen resolution.
- Set to Max to try to use largest available screen resolution.

On HiDPI screens APPLE_VENDOR_VARIABLE_GUID UIScale NVRAM variable may need to be set to 02 to enable HiDPI scaling in FileVault 2 UEFI password interface and boot screen logo. Refer to Recommended Variables section for more details.

Note: This will fail when console handle has no GOP protocol. When the firmware does not provide it, it can be added with ProvideConsoleGop UEFI quirk set to true.

8. ShowPicker

Type: plist boolean Failsafe: false

Description: Show simple boot picker to allow boot entry selection.

9. Timeout

Type: plist integer, 32 bit

Failsafe: 0

Description: Timeout in seconds in boot picker before automatic booting of the default boot entry. Use 0 to disable timer.

10. UsePicker

Type: plist boolean

Failsafe: false

Description: Use OpenCore built-in boot picker for boot management.

UsePicker set to false entirely disables all boot management in OpenCore except policy enforcement. In this case a custom user interface may utilise OcSupportPkg OcBootManagementLib to implement a user friendly boot picker oneself. Reference example of external graphics interface is provided in ExternalUi test driver.

OpenCore built-in boot picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and currently consists of the following options:

- Default this is the default option, and it lets OpenCore built-in boot picker to loads the default boot option as specified in Startup Disk preference pane.
- ShowPicker this option forces picker to show. Normally it can be achieved by holding OPT key during boot. Setting ShowPicker to true will make ShowPicker the default option.
- ResetNvram this option performs select UEFI variable erase and is normally achieved by holding CMD+OPT+P+R key combination during boot. Another way to erase UEFI variables is to choose Reset NVRAM in the picker. This option requires AllowNvramReset to be set to true.
- BootApple this options performs booting to the first found Apple operating system unless the default chosen operating system is already made by Apple. Hold X key to choose this option.
- BootAppleRecovery this option performs booting to Apple operating system recovery. Either the one related to the default chosen operating system, or first found in case default chosen operating system is not made by Apple or has no recovery. Hold CMD+R key combination to choose this option.

Note: activated KeySupport, UsbKbDxe, or similar driver is required for key handling to work. On many firmwares it is not possible to get all the keys function.

In addition to OPT OpenCore supports Escape key ShowPicker. This key exists for firmwares with PS/2 keyboards that fail to report held OPT key and require continual presses of Escape key to enter the boot menu.

8.4 Debug Properties

1. DisableWatchDog

 \mathbf{Type} : plist boolean

Failsafe: false

Description: Select firmwares may not succeed in quickly booting the operating system, especially in debug mode, which results in watch dog timer aborting the process. This option turns off watch dog timer.

2. DisplayDelay

Type: plist integer

Failsafe: 0

Description: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

3. DisplayLevel

Type: plist integer, 64 bit

Failsafe: 0

Description: EDK II debug level bitmask (sum) showed onscreen. Unless Target enables console (onscreen) printing, onscreen debug output will not be visible. The following levels are supported (discover more in DebugLib.h):

- 0x00000002 (bit 1) DEBUG_WARN in DEBUG, NOOPT, RELEASE.
- 0x00000040 (bit 6) DEBUG_INFO in DEBUG, NOOPT.
- 0x00400000 (bit 22) DEBUG_VERBOSE in custom builds.
- 0x80000000 (bit 31) DEBUG_ERROR in DEBUG, NOOPT, RELEASE.

4. Target

Type: plist integer

Failsafe: 0

Description: A bitmask (sum) of enabled logging targets. By default all the logging output is hidden, so this option is required to be set when debugging is necessary.

The following logging targets are supported:

- 0x01 (bit 0) Enable logging, otherwise all log is discarded.
- 0x02 (bit 1) Enable basic console (onscreen) logging.
- 0x04 (bit 2) Enable logging to Data Hub.
- 0x08 (bit 3) Enable serial port logging.
- 0x10 (bit 4) Enable UEFI variable logging.
- 0x20 (bit 5) Enable non-volatile UEFI variable logging.
- 0x40 (bit 6) Enable logging to file.

Console logging prints less than all the other variants. Depending on the build type (RELEASE, DEBUG, or NOOPT) different amount of logging may be read (from least to most).

Data Hub log will not log kernel and kext patches. To obtain Data Hub log use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree | grep boot-log | sort | sed 's/.*<\(.*\)>.*/\1/' | xxd -r -p
```

UEFI variable log does not include some messages and has no performance data. For safety reasons log size is limited to 32 kilobytes. Some firmwares may truncate it much earlier or drop completely if they have no memory. Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-log |
awk '{gsub(/%0d%0a%00/,"");gsub(/%0d%0a/,"\n")}1'
```

Warning: Some firmwares are reported to have broken NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

While OpenCore boot log already contains basic version information with build type and date, this data may also be found in NVRAM in opencore-version variable even with boot log disabled.

File logging will create a file named opencore-YYYY-MM-DD-HHMMSS.txt at EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that DisableWatchDog is set to true when you use a slow drive.

8.5 Security Properties

1. AllowNvramReset

Type: plist boolean Failsafe: false

Description: Allow CMD+OPT+P+R handling and enable showing NVRAM Reset entry in boot picker.

$2. \ {\tt ExposeSensitiveData}$

Type: plist integer

Failsafe: 0x6

Description: Sensitive data exposure bitmask (sum) to operating system.

- 0x01 Expose printable booter path as an UEFI variable.
- 0x02 Expose OpenCore version as an UEFI variable.
- 0x04 Expose OpenCore version in boot picker menu title.

Exposed booter path points to OpenCore.efi or its booter depending on the load order. To obtain booter path use the following command in macOS:

nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path

To use booter path for mounting booter volume use the following command in macOS:

```
 u = (nvram \ 4D1FDA02 - 38C7 - 4A6A - 9CC6 - 4BCCA8B30102:boot-path \ | \ sed \ 's/.*GPT, \ ([^,]*\ ), .*/\ |'); \ | \ if \ ["$u" \ != ""]; \ then \ sudo \ diskutil \ mount \ $u \ ; \ fi
```

To obtain OpenCore version use the following command in macOS:

nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:opencore-version

3. HaltLevel

Type: plist integer, 64 bit

Failsafe: 0x80000000 (DEBUG_ERROR)

Description: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of HaltLevel. Possible values match DisplayLevel values.

4. RequireSignature

Type: plist boolean

Failsafe: true

Description: Require vault.sig signature file for vault.plist in OC directory.

This file should contain a raw 256 byte RSA-2048 signature from SHA-256 hash of vault.plist. The signature is verified against the public key embedded into OpenCore.efi.

To embed the public key you should do either of the following:

- Provide public key during the OpenCore.efi compilation in OpenCoreVault.c file.
- Binary patch OpenCore.efi replacing zeroes with the public key between =BEGIN OC VAULT= and ==END OC VAULT== ASCII markers.

RSA public key 520 byte format description can be found in Chromium OS documentation. To convert public key from X.509 certificate or from PEM file use RsaTool.

Note: vault.sig is used regardless of this option when public key is embedded into OpenCore.efi. Setting it to true will only ensure configuration sanity, and abort the boot process when public key is not set but was supposed to be used for verification.

5. RequireVault

Type: plist boolean

Failsafe: true

Description: Require vault.plist file present in OC directory.

This file should contain SHA-256 hashes for all files used by OpenCore. Presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not happen unnoticed. To create this file automatically use create_vault.sh script.

Regardless of the underlying filesystem, path name and case must match between config.plist and vault.plist.

Note: vault.plist is tried to be read regardless of the value of this option, but setting it to true will ensure configuration sanity, and abort the boot process.

The complete set of commands to:

- Create vault.plist.
- Create a new RSA key (always do this to avoid loading old configuration).
- Embed RSA key into OpenCore.efi.
- Create vault.sig.

Can look as follows:

```
cd /Volumes/EFI/EFI/OC
/path/to/create_vault.sh .
/path/to/RsaTool -sign vault.plist vault.sig vault.pub
off=$(($(strings -a -t d OpenCore.efi | grep "=BEGIN OC VAULT=" | cut -f1 -d' ')+16))
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=528 conv=notrunc
rm vault.pub
```

Note: While it may appear obvious, but you have to use an external method to verify OpenCore.efi and BOOTx64.efi for secure boot path. For this you are recommended to at least enable UEFI SecureBoot with a custom certificate, and sign OpenCore.efi and BOOTx64.efi with your custom key. More details on customising secure boot on modern firmwares can be found in Taming UEFI SecureBoot paper (in Russian).

6. ScanPolicy

Type: plist integer, 32 bit

Failsafe: 0xF0103

Description: Define operating system detection policy.

This value allows to prevent scanning (and booting) from untrusted source based on a bitmask (sum) of select flags. As it is not possible to reliably detect every file system or device type, this feature cannot be fully relied upon in open environments, and the additional measures are to be applied.

Third party drivers may introduce additional security (and performance) measures following the provided scan policy. Scan policy is exposed in scan-policy variable of 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102 GUID for UEFI Boot Services only.

- 0x00000001 (bit 0) OC_SCAN_FILE_SYSTEM_LOCK, restricts scanning to only known file systems defined as
 a part of this policy. File system drivers may not be aware of this policy, and to avoid mounting of undesired
 file systems it is best not to load its driver. This bit does not affect dmg mounting, which may have any file
 system. Known file systems are prefixed with OC_SCAN_ALLOW_FS_.
- 0x00000002 (bit 1) 0C_SCAN_DEVICE_LOCK, restricts scanning to only known device types defined as a part of this policy. This is not always possible to detect protocol tunneling, so be aware that on some systems it may be possible for e.g. USB HDDs to be recognised as SATA. Cases like this must be reported. Known device types are prefixed with OC_SCAN_ALLOW_DEVICE_.
- 0x00000100 (bit 8) OC_SCAN_ALLOW_FS_APFS, allows scanning of APFS file system.
- 0x00000200 (bit 9) OC_SCAN_ALLOW_FS_HFS, allows scanning of HFS file system.
- 0x00000400 (bit 10) 0C_SCAN_ALLOW_FS_ESP, allows scanning of EFI System Partition file system.
- 0x00000800 (bit 11) 0C_SCAN_ALLOW_FS_NTFS, allows scanning of NTFS (Msft Basic Data) file system.
- 0x00001000 (bit 12) 0C_SCAN_ALLOW_FS_EXT, allows scanning of EXT (Linux Root) file system.
- 0x00010000 (bit 16) OC_SCAN_ALLOW_DEVICE_SATA, allow scanning SATA devices.
- 0x00020000 (bit 17) 0C_SCAN_ALLOW_DEVICE_SASEX, allow scanning SAS and Mac NVMe devices.

- 0x00040000 (bit 18) OC_SCAN_ALLOW_DEVICE_SCSI, allow scanning SCSI devices.
- 0x00080000 (bit 19) OC_SCAN_ALLOW_DEVICE_NVME, allow scanning NVMe devices.
- 0x00100000 (bit 20) OC_SCAN_ALLOW_DEVICE_ATAPI, allow scanning CD/DVD devices.
- 0x00200000 (bit 21) OC_SCAN_ALLOW_DEVICE_USB, allow scanning USB devices.
- 0x00400000 (bit 22) 0C_SCAN_ALLOW_DEVICE_FIREWIRE, allow scanning FireWire devices.
- 0x00800000 (bit 23) OC_SCAN_ALLOW_DEVICE_SDCARD, allow scanning card reader devices.

Note: Given the above description, 0xF0103 value is expected to allow scanning of SATA, SAS, SCSI, and NVMe devices with APFS file system, and prevent scanning of any devices with HFS or FAT32 file systems in addition to not scanning APFS file systems on USB, CD, and FireWire drives. The combination reads as:

- OC_SCAN_FILE_SYSTEM_LOCK
- OC_SCAN_DEVICE_LOCK
- OC SCAN ALLOW FS APFS
- OC_SCAN_ALLOW_DEVICE_SATA
- OC_SCAN_ALLOW_DEVICE_SASEX
- OC_SCAN_ALLOW_DEVICE_SCSI
- OC_SCAN_ALLOW_DEVICE_NVME

8.6 Entry Properties

1. Arguments

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.

2. Comment

Type: plist string Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. Enabled

Type: plist boolean Failsafe: false

Description: This entry will not be listed unless set to true.

4. Name

Type: plist string Failsafe: Empty string

Description: Human readable entry name displayed in boot picker.

5 Path

Type: plist string Failsafe: Empty string

 $\textbf{Description} \hbox{: } \textbf{Entry location depending on entry type.}$

- Entries specify external boot options, and therefore take device paths in Path key. These values are not checked, thus be extremely careful. Example: PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI
- Tools specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to OC/Tools directory. Example: Shell.efi.

9 NVRAM

9.1 Introduction

Has plist dict type and allows to set volatile UEFI variables commonly referred as NVRAM variables. Refer to man nvram for more details. macOS extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication, and thus supplying several NVRAM is required for proper macOS functioning.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' NVRAM variable belongs to. macOS uses several GUIDs, including but not limited to:

- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14 (APPLE_VENDOR_VARIABLE_GUID)
- 7C436110-AB2A-4BBB-A880-FE41995C9F82 (APPLE_BOOT_VARIABLE_GUID)
- 8BE4DF61-93CA-11D2-AAOD-00E098032B8C (EFI_GLOBAL_VARIABLE_GUID)
- 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102 (OC_VENDOR_VARIABLE_GUID)

Note: Some of the variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Please ensure that variables of this section never collide with them, as behaviour is undefined otherwise.

9.2 Properties

1. Add

Type: plist dict

Description: Sets NVRAM variables from a map (plist dict) of GUIDs to a map (plist dict) of variable names and their values in plist metadata format. GUIDs must be provided in canonic string format in upper or lower case (e.g. 8BE4DF61-93CA-11D2-AAOD-00E098032B8C).

Created variables get EFI_VARIABLE_BOOTSERVICE_ACCESS and EFI_VARIABLE_RUNTIME_ACCESS attributes set. Variables will only be set if not present and not blocked. To overwrite a variable add it to Block section. This approach enables to provide default values till the operating system takes the lead.

Note: If plist key does not conform to GUID format, behaviour is undefined.

$2.\ {\tt Block}$

Type: plist dict

Description: Removes NVRAM variables from a map (plist dict) of GUIDs to an array (plist array) of variable names in plist string format.

3. LegacyEnable

Type: plist boolean Failsafe: false

Description: Enables loading of NVRAM variable file named nvram.plist from EFI volume root.

This file must have root plist dictionary type and contain two fields:

- Version plist integer, file version, must be set to 1.
- Add plist dictionary, equivalent to Add from config.plist.

Variable loading happens prior to Block (and Add) phases, and will not overwrite any existing variable. Variables allowed to be set must be specified in LegacySchema. Third-party scripts may be used to create nvram.plist file. An example of such script can be found in Utilities. The use of third-party scripts may require ExposeSensitiveData set to 0x3 to provide boot-path variable with OpenCore EFI partition UUID.

WARNING: This feature is very dangerous as it passes unprotected data to your firmware variable services. Use it only when no hardware NVRAM implementation is provided by the firmware or it is incompatible.

4. LegacySchema

Type: plist dict

Description: Allows setting select NVRAM variables from a map (plist dict) of GUIDs to an array (plist array) of variable names in plist string format.

You can use * value to accept all variables for select GUID.

WARNING: Choose variables very carefully, as nvram.plist is not vaulted. For instance, do not put boot-args or csr-active-config, as this can bypass SIP.

To read NVRAM variable value from macOS one could use nvram by concatenating variable GUID and name separated by: symbol. For example, nvram 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args.

A continuously updated variable list can be found in a corresponding document: NVRAM Variables.

9.3 Mandatory Variables

Warning: These variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Using PlatformInfo is the recommend way of setting these variables.

The following variables are mandatory for macOS functioning:

- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures 32-bit FirmwareFeatures. Present on all Macs to avoid extra parsing of SMBIOS tables
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask 32-bit FirmwareFeaturesMask. Present on all Macs to avoid extra parsing of SMBIOS tables.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB
 BoardSerialNumber. Present on newer Macs (2013+ at least) to avoid extra parsing of SMBIOS tables, especially in boot.efi.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM
 Primary network adapter MAC address or replacement value. Present on newer Macs (2013+ at least) to avoid accessing special memory region, especially in boot.efi.

9.4 Recommended Variables

The following variables are recommended for faster startup or other improvements:

- 7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config 32-bit System Integrity Protection bitmask. Declared in XNU source code in csr.h.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14: ExtendedFirmwareFeatures
 Combined FirmwareFeatures and ExtendedFirmwareFeatures. Present on newer Macs to avoid extra parsing of SMBIOS tables
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask

 Combined FirmwareFeaturesMask and ExtendedFirmwareFeaturesMask. Present on newer Macs to avoid extra parsing of SMBIOS tables.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID Hardware BoardProduct (e.g. Mac-35C1E88140C3E6CF). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in boot.efi.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB Hardware BoardSerialNumber. Override for MLB. Present on newer Macs (2013+ at least).
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd
 ASCII string defining default keyboard layout. Format is lang-COUNTRY:keyboard, e.g. ru-RU:252 for Russian locale and ABC keyboard. Also accepts short forms: ru:252 or ru:0 (U.S. keyboard, compatible with 10.9). Full decoded keyboard list from AppleKeyboardLayouts-L.dat can be found here. Using non-latin keyboard on 10.14 will not enable ABC keyboard, unlike previous and subsequent macOS versions, and is thus not recommended in case you need 10.14.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:security-mode

 ASCII string defining FireWire security mode. Legacy, can be found in IOFireWireFamily source code in IOFireWireController.cpp. It is recommended not to set this variable, which may speedup system startup. Setting to full is equivalent to not setting the variable and none disables FireWire security.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale
 One-byte data defining boot.efi user interface scaling. Should be 01 for normal screens and 02 for HiDPI screens.

9.5 Other Variables

The following variables may be useful for certain configurations or troubleshooting:

• 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args

Kernel arguments, used to pass configuration to Apple kernel and drivers. There are many arguments, which may be found by looking for the use of PE_parse_boot_argn function in the kernel or driver code. Some of the known boot arguments include:

- acpi_layer=0xFFFFFFF
- acpi_level=0xFFFF5F (implies ACPI_ALL_COMPONENTS)
- batman=VALUE (AppleSmartBatteryManager debug mask)
- batman-nosmc=1 (disable AppleSmartBatteryManager SMC interface)
- cpus=VALUE (maximum number of CPUs used)
- debug=VALUE (debug mask)
- io=VALUE (IOKit debug mask)
- keepsyms=1 (show panic log debug symbols)
- kextlog=VALUE (kernel extension loading debug mask)
- nv_disable=1 (disables NVIDIA GPU acceleration)
- nvda_drv=1 (legacy way to enable NVIDIA web driver, removed in 10.12)
- npci=0x2000 (legacy, disables kIOPCIConfiguratorPFM64)
- lapic_dont_panic=1
- slide=VALUE (manually set KASLR slide)
- smcdebug=VALUE (AppleSMC debug mask)
- -amd_no_dgpu_accel (alternative to WhateverGreen's -radvesa for new GPUs)
- -nehalem_error_disable
- -no_compat_check (disable model checking)
- -s (single mode)
- v (verbose mode)
- -x (safe mode)

There are multiple external places summarising macOS argument lists: example 1, example 2.

• 7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg

Booter arguments, similar to boot-args but for boot.efi. Accepts a set of arguments, which are hexadecimal 64-bit values with or without 0x prefix primarily for logging control:

- log=VALUE
 - * 1 AppleLoggingConOutOrErrSet/AppleLoggingConOutOrErrPrint (classical ConOut/StdErr)
 - * 2 AppleLoggingStdErrSet/AppleLoggingStdErrPrint (StdErr or serial?)
 - * 4 AppleLoggingFileSet/AppleLoggingFilePrint (BOOTER.LOG/BOOTER.OLD file on EFI partition)
- debug=VALUE
 - * 1 enables print something to BOOTER.LOG (stripped code implies there may be a crash)
 - * 2 enables perf logging to /efi/debug-log in the device three
 - * 4 enables timestamp printing for styled printf calls
- level=VALUE Verbosity level of DEBUG output. Everything but 0x80000000 is stripped from the binary, and this is the default value.
- kc-read-size=VALUE Chunk size used for buffered I/O from network or disk for prelinkedkernel reading and related. Set to 1MB (0x100000) by default, can be tuned for faster booting.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once

Booter arguments override removed after first launch. Otherwise equivalent to bootercfg.

- 7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name
 - Current saved host name. ASCII string.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv

NVIDIA Web Driver control variable. Takes ASCII digit 1 or 0 to enable or disable installed driver.

PlatformInfo 10

Platform information is comprised of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from MacInfoPkg package, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

Most of the fields specify the overrides in SMBIOS, and their field names conform to EDK2 SmBios.h header file. However, several important fields reside in Data Hub and NVRAM. Some of the values can be found in more than one field and/or destination, so there are two ways to control their update process: manual, where one specifies all the values (the default), and semi-automatic, where (Automatic) only select values are specified, and later used for system configuration.

To inspect SMBIOS contents dmidecode utility can be used. Version with macOS specific enhancements can be downloaded from Acidanthera/dmidecode.

10.1**Properties**

1. Automatic

Type: plist boolean Failsafe: false

Description: Generate PlatformInfo based on Generic section instead of using values from DataHub, NVRAM, and SMBIOS sections.

Enabling this option is useful when Generic section is flexible enough. When enabled SMBIOS, DataHub, and PlatformNVRAM data is unused.

2. UpdateDataHub

Type: plist boolean Failsafe: false

Description: Update Data Hub fields. These fields are read from Generic or DataHub sections depending on Automatic value.

3. UpdateNVRAM

Type: plist boolean Failsafe: false

Description: Update NVRAM fields related to platform information.

These fields are read from Generic or PlatformNVRAM sections depending on Automatic value. All the other fields are to be specified with NVRAM section.

If UpdateNVRAM is set to false the aforementioned variables can be updated with NVRAM section. If UpdateNVRAM is set to true the behaviour is undefined when any of the fields are present in NVRAM section.

4. UpdateSMBIOS

Type: plist boolean Failsafe: false

Description: Update SMBIOS fields. These fields are read from Generic or SMBIOS sections depending on Automatic value.

5. UpdateSMBIOSMode

Type: plist string Failsafe: Create

Description: Update SMBIOS fields approach:

- TryOverwrite Overwrite if new size is <= than the page-aligned original and there are no issues with legacy region unlock. Create otherwise. Has issues with some firmwares.
- Create Replace the tables with newly allocated EfiReservedMemoryType at AllocateMaxAddress without any fallbacks.

- Overwrite Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- Custom Write first SMBIOS table (gEfiSmbiosTableGuid) to gOcCustomSmbiosTableGuid to workaround firmwares overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to Create. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: "EB9D2D31" "EB9D2D35" (in ASCII), done automatically by CustomSMBIOSGuid quirk.
- 6. Generic

Type: plist dictonary

Optional: When Automatic is false

Description: Update all fields. This section is read only when Automatic is active.

7. DataHub

Type: plist dictonary

Optional: When Automatic is true

Description: Update Data Hub fields. This section is read only when Automatic is not active.

8. PlatformNVRAM

Type: plist dictonary

Optional: When Automatic is true

Description: Update platform NVRAM fields. This section is read only when Automatic is not active.

9. SMBIOS

Type: plist dictonary

Optional: When Automatic is true

Description: Update SMBIOS fields. This section is read only when Automatic is not active.

10.2 Generic Properties

1. SpoofVendor

Type: plist boolean

Failsafe: false

Description: Sets SMBIOS vendor fields to Acidanthera.

It is dangerous to use Apple in SMBIOS vendor fields for reasons given in SystemManufacturer description. However, certain firmwares may not provide valid values otherwise, which could break some software.

2. SystemProductName

Type: plist string Failsafe: MacPro6,1

Description: Refer to SMBIOS SystemProductName.

3. SystemSerialNumber

Type: plist string Failsafe: OPENCORE_SN1

 ${\bf Description} \hbox{: Refer to SMBIOS SystemSerialNumber}.$

4. SystemUUID

Type: plist string, GUID Failsafe: OEM specified

Description: Refer to SMBIOS SystemUUID.

5. MLB

Type: plist string

Failsafe: OPENCORE_MLB_SN11

Description: Refer to SMBIOS BoardSerialNumber.

6. ROM

Type: plist data, 6 bytes

Failsafe: all zero

 $\textbf{Description} \colon \operatorname{Refer} \ \text{to} \ 4 \text{D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM}.$

10.3 DataHub Properties

1. PlatformName

Type: plist string Failsafe: Not installed

Description: Sets name in gEfiMiscSubClassGuid. Value found on Macs is platform in ASCII.

2. SystemProductName

Type: plist string Failsafe: Not installed

 $\textbf{Description: Sets Model in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS \ \textbf{SystemProductName}$

in Unicode.

3. SystemSerialNumber

Type: plist string Failsafe: Not installed

Description: Sets SystemSerialNumber in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS

SystemSerialNumber in Unicode.

4. SystemUUID

Type: plist string, GUID Failsafe: Not installed

Description: Sets system-id in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS SystemUUID.

5. BoardProduct

Type: plist string Failsafe: Not installed

Description: Sets board-id in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS BoardProduct

in ASCII.

6. BoardRevision

Type: plist data, 1 byte

Failsafe: 0

Description: Sets board-rev in gEfiMiscSubClassGuid. Value found on Macs seems to correspond to internal board revision (e.g. 01).

7. StartupPowerEvents

Type: plist integer, 64-bit

Failsafe: 0

Description: Sets StartupPowerEvents in gEfiMiscSubClassGuid. Value found on Macs is power management state bitmask, normally 0. Known bits read by X86PlatformPlugin.kext:

- 0x00000001 Shutdown cause was a PWROK event (Same as GEN_PMCON_2 bit 0)
- 0x00000002 Shutdown cause was a SYS PWROK event (Same as GEN PMCON 2 bit 1)
- 0x00000004 Shutdown cause was a THRMTRIP# event (Same as GEN PMCON 2 bit 3)
- 0x00000008 Rebooted due to a SYS RESET# event (Same as GEN_PMCON_2 bit 4)
- 0x00000010 Power Failure (Same as GEN_PMCON_3 bit 1 PWR_FLR)
- 0x00000020 Loss of RTC Well Power (Same as GEN_PMCON_3 bit 2 RTC_PWR_STS)
- 0x00000040 General Reset Status (Same as GEN_PMCON_3 bit 9 GEN_RST_STS)
- Oxffffff80 SUS Well Power Loss (Same as GEN_PMCON_3 bit 14)
- 0x00010000 Wake cause was a ME Wake event (Same as PRSTS bit 0, ME_WAKE_STS)
- 0x00020000 Cold Reboot was ME Induced event (Same as PRSTS bit 1 ME_HRST_COLD_STS)
- 0x00040000 Warm Reboot was ME Induced event (Same as PRSTS bit 2 ME_HRST_WARM_STS)
- 0x00080000 Shutdown was ME Induced event (Same as PRSTS bit 3 ME_HOST_PWRDN)
- 0x00100000 Global reset ME Wachdog Timer event (Same as PRSTS bit 6)
- 0x00200000 Global reset PowerManagment Wachdog Timer event (Same as PRSTS bit 15)

8. InitialTSC

Type: plist integer, 64-bit

Failsafe: 0

Description: Sets Initial TSC in gEfiProcessor SubClass Guid. Sets initial TSC value, normally 0.

9. FSBFrequency

Type: plist integer, 64-bit

Failsafe: Automatic

Description: Sets FSBFrequency in gEfiProcessorSubClassGuid.

Sets CPU FSB frequency. This value equals to CPU nominal frequency divided by CPU maximum bus ratio and is specified in Hz. Refer to MSR_NEHALEM_PLATFORM_INFO (CEh) MSR value to determine maximum bus ratio on modern Intel CPUs.

Note: This value is not used on Skylake and newer but is still provided to follow suit.

10. ARTFrequency

Type: plist integer, 64-bit

Failsafe: Automatic

Description: Sets ARTFrequency in gEfiProcessorSubClassGuid.

This value contains CPU ART frequency, also known as crystal clock frequency. Its existence is exclusive to Skylake generation and newer. The value is specified in Hz, and is normally 24 MHz for client Intel segment, 25 MHz for server Intel segment, and 19.2 MHz for Intel Atom CPUs. macOS till 10.15 inclusive assumes 24 MHz by default.

Note: On Intel Skylake X ART frequency may be a little less (approx. 0.25%) than 24 or 25 MHz due to special EMI-reduction circuit as described in Acidanthera Bugtracker.

11. DevicePathsSupported

Type: plist integer, 32-bit

Failsafe: Not installed

Description: Sets DevicePathsSupported in gEfiMiscSubClassGuid. Must be set to 1 for AppleACPIPlatform.kext to append SATA device paths to Boot#### and efi-boot-device-data variables. Set to 1 on all modern Macs.

12. SmcRevision

Type: plist data, 6 bytes Failsafe: Not installed

Description: Sets REV in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC REV key.

13. SmcBranch

Type: plist data, 8 bytes Failsafe: Not installed

Description: Sets RBr in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC RBr kev.

14. SmcPlatform

Type: plist data, 8 bytes Failsafe: Not installed

Description: Sets RPlt in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to

generate SMC RPlt key.

10.4 PlatformNVRAM Properties

1. BID

Type: plist string Failsafe: Not installed

Description: Specifies the value of NVRAM variable 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID.

2. ROM

Type: plist data, 6 bytes Failsafe: Not installed

Description: Specifies the values of NVRAM variables $4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM$ and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM.

3. MLB

Type: plist string Failsafe: Not installed

 $\textbf{Description}: \ \textbf{Specifies the values of NVRAM variables 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB \ and \$

4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB.

4. FirmwareFeatures

Type: plist data, 8 bytes Failsafe: Not installed

Description: This variable comes in pair with FirmwareFeaturesMask. Specifies the values of NVRAM variables:

• 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures

• 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures

5. FirmwareFeaturesMask

Type: plist data, 8 bytes Failsafe: Not installed

Description: This variable comes in pair with FirmwareFeatures. Specifies the values of NVRAM variables:

4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask

• 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask

10.5 SMBIOS Properties

1. BIOSVendor

Type: plist string Failsafe: OEM specified

SMBIOS: BIOS Information (Type 0) — Vendor

Description: BIOS Vendor. All rules of SystemManufacturer do apply.

2. BIOSVersion

Type: plist string Failsafe: OEM specified

SMBIOS: BIOS Information (Type 0) — BIOS Version

Description: Firmware version. This value gets updated and takes part in update delivery configuration and macOS version compatibility. This value could look like MM71.88Z.0234.B00.1809171422 in older firmwares, and is described in BiosId.h. In newer firmwares it should look like 236.0.0.0.0 or 220.230.16.0.0 (iBridge: 16.16.2542.0.0,0). iBridge version is read from BridgeOSVersion variable, and is only present on macs with T2.

Apple ROM Version

BIOS ID: MBP151.88Z.F000.B00.1811142212

Model: MBP151

EFI Version: 220.230.16.0.0 Built by: root@quinoa

Date: Wed Nov 14 22:12:53 2018

Revision: 220.230.16 (B&I)

ROM Version: F000 B00

Build Type: Official Build, RELEASE

Compiler: Apple LLVM version 10.0.0 (clang-1000.2.42)

UUID: E5D1475B-29FF-32BA-8552-682622BA42E1 UUID: 151B0907-10F9-3271-87CD-4BF5DBECACF5

3. BIOSReleaseDate

Type: plist string Failsafe: OEM specified

SMBIOS: BIOS Information (Type 0) — BIOS Release Date

Description: Firmware release date. Similar to BIOSVersion. May look like 12/08/2017.

 $4. \ {\tt SystemManufacturer}$

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1) — Manufacturer

Description: OEM manufacturer of the particular board. Shall not be specified unless strictly required. Should *not* contain Apple Inc., as this confuses numerous services present in the operating system, such as firmware updates, eficheck, as well as kernel extensions developed in Acidanthera, such as Lilu and its plugins. In addition it will also make some operating systems like Linux unbootable.

$5. \ {\tt SystemProductName}$

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1), Product Name

Description: Preferred Mac model used to mark the device as supported by the operating system. This value must be specified by any configuration for later automatic generation of the related values in this and other SMBIOS tables and related configuration parameters. If <code>SystemProductName</code> is not compatible with the target operating system, <code>-no_compat_check</code> boot argument may be used as an override.

Note: If SystemProductName is unknown, and related fields are unspecified, default values should be assumed as being set to MacPro6,1 data. The list of known products can be found in MacInfoPkg.

6. SystemVersion

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1) — Version

Description: Product iteration version number. May look like 1.1.

7. SystemSerialNumber

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1) — Serial Number

Description: Product serial number in defined format. Known formats are described in macserial.

8. SystemUUID

Type: plist string, GUID Failsafe: OEM specified

SMBIOS: System Information (Type 1) — UUID

Description: A UUID is an identifier that is designed to be unique across both time and space. It requires no central registration process.

9. SystemSKUNumber

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1) — SKU Number

Description: Mac Board ID (board-id). May look like Mac-7BA5B2D9E42DDD94 or Mac-F221BEC8 in older models. Sometimes it can be just empty.

10. SystemFamily

Type: plist string Failsafe: OEM specified

SMBIOS: System Information (Type 1) — Family **Description**: Family name. May look like iMac Pro.

11. BoardManufacturer

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) - Manufacturer

Description: Board manufacturer. All rules of SystemManufacturer do apply.

12. BoardProduct

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) - Product

Description: Mac Board ID (board-id). May look like Mac-7BA5B2D9E42DDD94 or Mac-F221BEC8 in older

models.

13. BoardVersion

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) - Version

Description: Board version number. Varies, may match SystemProductName or SystemProductVersion.

14. BoardSerialNumber

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) — Serial Number

Description: Board serial number in defined format. Known formats are described in macserial.

15. BoardAssetTag

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) — Asset Tag

Description: Asset tag number. Varies, may be empty or Type2 - Board Asset Tag.

16. BoardType

Type: plist integer Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) — Board Type

Description: Either OxA (Motherboard (includes processor, memory, and I/O) or OxB (Processor/Memory

Module), refer to Table 15 – Baseboard: Board Type for more details.

17. BoardLocationInChassis

Type: plist string Failsafe: OEM specified

SMBIOS: Baseboard (or Module) Information (Type 2) — Location in Chassis

Description: Varies, may be empty or Part Component.

18. ChassisManufacturer

Type: plist string Failsafe: OEM specified

SMBIOS: System Enclosure or Chassis (Type 3) — Manufacturer

Description: Board manufacturer. All rules of SystemManufacturer do apply.

19. ChassisType

Type: plist integer Failsafe: OEM specified

SMBIOS: System Enclosure or Chassis (Type 3) — Type

Description: Chassis type, refer to Table 17 — System Enclosure or Chassis Types for more details.

20. ChassisVersion

Type: plist string Failsafe: OEM specified

SMBIOS: System Enclosure or Chassis (Type 3) — Version

Description: Should match BoardProduct.

21. ChassisSerialNumber

Type: plist string Failsafe: OEM specified

SMBIOS: System Enclosure or Chassis (Type 3) — Version

Description: Should match SystemSerialNumber.

22. ChassisAssetTag

Type: plist string Failsafe: OEM specified

SMBIOS: System Enclosure or Chassis (Type 3) — Asset Tag Number

Description: Chassis type name. Varies, could be empty or MacBook-Aluminum.

23. PlatformFeature

Type: plist integer, 32-bit

Failsafe: 0xFFFFFFF

SMBIOS: APPLE_SMBIOS_TABLE_TYPE133 - PlatformFeature

Description: Platform features bitmask. Refer to AppleFeatures.h for more details. Use Oxffffffff value to

not provide this table.

24. SmcVersion

Type: plist data, 16 bytes

Failsafe: All zero

SMBIOS: APPLE_SMBIOS_TABLE_TYPE134 - Version

Description: ASCII string containing SMC version in upper case. Missing on T2 based Macs. Ignored when

zero.

25. FirmwareFeatures

Type: plist data, 8 bytes

Failsafe: 0

 ${\bf SMBIOS: APPLE_SMBIOS_TABLE_TYPE128 - Firmware Features} \ {\bf and} \ {\bf ExtendedFirmware Features}$

Description: 64-bit firmware features bitmask. Refer to AppleFeatures.h for more details. Lower 32 bits match

FirmwareFeatures. Upper 64 bits match ExtendedFirmwareFeatures.

26. FirmwareFeaturesMask

Type: plist data, 8 bytes

Failsafe: 0

SMBIOS: APPLE_SMBIOS_TABLE_TYPE128 - FirmwareFeaturesMask and ExtendedFirmwareFeaturesMask

Description: Supported bits of extended firmware features bitmask. Refer to AppleFeatures.h for more details.

Lower 32 bits match FirmwareFeaturesMask. Upper 64 bits match ExtendedFirmwareFeaturesMask.

27. ProcessorType

Type: plist integer, 16-bit

Failsafe: Automatic

SMBIOS: APPLE_SMBIOS_TABLE_TYPE131 - ProcessorType **Description**: Combined of Processor Major and Minor types.

28. MemoryFormFactor

Type: plist integer, 8-bit Failsafe: OEM specified

SMBIOS: Memory Device (Type 17) — Form Factor

Description: Memory form factor. On Macs it should be DIMM or SODIMM.

11 UEFI

11.1 Introduction

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows to load additional UEFI modules and/or apply tweaks for the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

11.2 Properties

1. ConnectDrivers

Type: plist boolean

Failsafe: false

Description: Perform UEFI controller connection after driver loading. This option is useful for loading filesystem drivers, which usually follow UEFI driver model, and may not start by themselves. While effective, this option is not necessary with e.g. APFS loader driver, and may slightly slowdown the boot.

2. Drivers

Type: plist array Failsafe: None

Description: Load selected drivers from OC/Drivers directory.

Designed to be filled with string filenames meant to be loaded as UEFI drivers. Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead your system to unbootable state or even cause permanent firmware damage. Some of the known drivers include:

- ApfsDriverLoader APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.
- FwRuntimeServices OC_FIRMWARE_RUNTIME protocol implementation that increases the security of Open-Core and Lilu by supporting read-only and write-only NVRAM variables. Some quirks, like RequestBootVarRouting, require this driver for proper function. Due to the nature of being a runtime driver, i.e. functioning in parallel with the target operating system, it cannot be implemented within OpenCore itself.
- EnhancedFatDxe FAT filesystem driver from FatPkg. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
- NvmExpressDxe NVMe support driver from MdeModulePkg. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
- UsbKbDxe USB keyboard driver adding the support of AppleKeyMapAggregator protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin KeySupport, which may work better or worse depending on the firmware.
- VirtualSmc UEFI SMC driver, required for proper FileVault 2 functionality and potentially other macOS specifics. An alternative, named SMCHelper, is not compatible with VirtualSmc and OpenCore, which is unaware of its specific interfaces. In case FakeSMC kernel extension is used, manual NVRAM variable addition may be needed and VirtualSmc driver should still be used.
- VBoxHfs HFS file system driver with bless support. This driver is an alternative to a closed source HFSPlus driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
- XhciDxe XHCI USB controller support driver from MdeModulePkg. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards.

To compile the drivers from UDK (EDK II) use the same command you do normally use for OpenCore compilation, but choose a corresponding package:

git clone https://github.com/acidanthera/audk UDK
cd UDK
source edksetup.sh

```
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p FatPkg/FatPkg.dsc
build -a X64 -b RELEASE -t XCODE5 -p MdeModulePkg/MdeModulePkg.dsc
```

3. Input

Type: plist dict Failsafe: None

Description: Apply individual settings designed for input (keyboard and mouse) in Input Properties section

below.

4. Protocols

Type: plist dict Failsafe: None

Description: Force builtin versions of select protocols described in Protocols Properties section below.

Note: all protocol instances are installed prior to driver loading.

5. Quirks

Type: plist dict Failsafe: None

Description: Apply individual firmware quirks described in Quirks Properties section below.

11.3 Input Properties

 $1. \ {\tt KeyForgetThreshold}$

Type: plist integer

Failsafe: 0

Description: Remove key unless it was submitted during this timeout in milliseconds.

AppleKeyMapAggregator protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

This option allows to set this timeout based on your platform. The recommended value that works on the majority of the platforms is 5 milliseconds. For reference, holding one key on VMware will repeat it roughly every 2 milliseconds and the same value for APTIO V is 3-4 milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

2. KeyMergeThreshold

Type: plist integer

Failsafe: 0

Description: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

Similarly to KeyForgetThreshold, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

Holding multiple keys results in reports every 2 and 1 milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least 6 and 10 milliseconds for the same platforms. The recommended value for this option is 2 milliseconds, but it may be decreased for faster platforms and increased for slower.

3. KeySupport

Type: plist boolean

Failsafe: false

Description: Enable internal keyboard input translation to AppleKeyMapAggregator protocol.

This option activates the internal keyboard interceptor driver, based on AppleGenericInput aka AptioIntputFix), to fill AppleKeyMapAggregator database for input functioning. In case a separate driver is used, such as UsbKbDxe, this option should never be enabled.

4. KeySupportMode

Type: plist string Failsafe: empty string

Description: Set internal keyboard input translation to AppleKeyMapAggregator protocol mode.

- Auto Performs automatic choice as available with the following preference: AMI, V2, V1.
- V1 Uses UEFI standard legacy input protocol EFI_SIMPLE_TEXT_INPUT_PROTOCOL.
- V2 Uses UEFI standard modern input protocol EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL.
- AMI Uses APTIO input protocol AMI_EFIKEYCODE_PROTOCOL.

5. KeySwap

Type: plist boolean Failsafe: false

Description: Swap Command and Option keys during submission.

This option may be useful for keyboard layouts with Option key situated to the right of Command key.

6. PointerSupport

Type: plist boolean Failsafe: false

Description: Enable internal pointer driver.

This option implements standard UEFI pointer protocol (EFI_SIMPLE_POINTER_PROTOCOL) through select OEM protocols. The option may be useful on Z87 ASUS boards, where EFI_SIMPLE_POINTER_PROTOCOL is broken.

7. PointerSupportMode

Type: plist string Failsafe: empty string

Description: Set OEM protocol used for internal pointer driver.

Currently the only supported variant is ASUS, using specialised protocol available on select Z87 and Z97 ASUS boards. More details can be found in LongSoft/UefiTool#116.

$8. \ {\tt TimerResolution}$

Type: plist integer

Failsafe: 0

Description: Set architecture timer resolution.

This option allows to update firmware architecture timer period with the specified value in 100 nanosecond units. Setting a lower value generally improves performance and responsiveness of the interface and input handling.

The recommended value is 50000 (5 milliseconds) or slightly higher. ASUS boards use 60000 for the interface. Apple boards use 100000.

11.4 Protocols Properties

1. AppleBootPolicy

Type: plist boolean

Failsafe: false

Description: Reinstalls Apple Boot Policy protocol with a builtin version. This may be used to ensure APFS compatibility on VMs or legacy Macs.

2. AppleEvent

Type: plist boolean

Failsafe: false

Description: Reinstalls Apple Event protocol with a builtin version. This may be used to ensure File Vault 2 compatibility on VMs or legacy Macs.

3. AppleImageConversion

Type: plist boolean

Failsafe: false

Description: Reinstalls Apple Image Conversion protocol with a builtin version.

4. AppleKeyMap

Type: plist boolean

Failsafe: false

Description: Reinstalls Apple Key Map protocols with builtin versions.

5. AppleUserInterfaceTheme Type: plist boolean

Failsafe: false

Description: Reinstalls Apple User Interface Theme protocol with a builtin version.

6. ConsoleControl

Type: plist boolean Failsafe: false

Description: Replaces Console Control protocol with a builtin version.

macOS bootloader requires console control protocol for text output, which some firmwares miss. This option is required to be set when the protocol is already available in the firmware, and other console control options are used, such as IgnoreTextInGraphics, SanitiseClearScreen, and sometimes ConsoleBehaviourOs with ConsoleBehaviourUi).

7. DataHub

Type: plist boolean

Failsafe: false

Description: Reinstalls Data Hub protocol with a builtin version. This will drop all previous properties if the protocol was already installed.

8. DeviceProperties

Type: plist boolean

Failsafe: false

Description: Reinstalls Device Property protocol with a builtin version. This will drop all previous properties if it was already installed. This may be used to ensure full compatibility on VMs or legacy Macs.

9. FirmwareVolume

Type: plist boolean Failsafe: false

Description: Forcibly wraps Firmware Volume protocols or installs new to support custom cursor images for File Vault 2. Should be set to true to ensure File Vault 2 compatibility on everything but VMs and legacy Macs.

10. HashServices

Type: plist boolean Failsafe: false

Description: Forcibly reinstalls Hash Services protocols with builtin versions. Should be set to true to ensure File Vault 2 compatibility on platforms providing broken SHA-1 hashing. Can be diagnosed by invalid cursor size with UIScale set to 02, in general platforms prior to APTIO V (Haswell and older) are affected.

11. UnicodeCollation

Type: plist boolean

Failsafe: false

Description: Forcibly reinstalls unicode collation services with builtin version. Should be set to true to ensure UEFI Shell compatibility on platforms providing broken unicode collation. In general legacy Insyde and APTIO platforms on Ivy Bridge and earlier are affected.

11.5Quirks Properties

1. AvoidHighAlloc

Type: plist boolean

Failsafe: false

Description: Advises allocators to avoid allocations above first 4 GBs of RAM.

This is a workaround for select board firmwares, namely GA-Z77P-D3 (rev. 1.1), failing to properly access higher memory in UEFI Boot Services. On these boards this quirk is required for booting entries that need to allocate large memory chunks, such as macOS DMG recovery entries. On unaffected boards it may cause boot failures, and thus strongly not recommended. For known issues refer to acidanthera/bugtracker#449.

2. ClearScreenOnModeSwitch

Type: plist boolean

Failsafe: false

Description: Some firmwares clear only part of screen when switching from graphics to text mode, leaving a fragment of previously drawn image visible. This option fills the entire graphics screen with black color before switching to text mode.

Note: ConsoleControl should be set to true for this to work.

3. ExitBootServicesDelay

Type: plist integer

Failsafe: 0

Description: Adds delay in microseconds after EXIT_BOOT_SERVICES event.

This is a very ugly quirk to circumvent "Still waiting for root device" message on select APTIO IV firmwares, namely ASUS Z87-Pro, when using FileVault 2 in particular. It seems that for some reason they execute code in parallel to EXIT_BOOT_SERVICES, which results in SATA controller being inaccessible from macOS. A better approach should be found in some future. Expect 3-5 seconds to be enough in case the quirk is needed.

4. IgnoreInvalidFlexRatio

Type: plist boolean

Failsafe: false

Description: Select firmwares, namely APTIO IV, may contain invalid values in MSR_FLEX_RATIO (0x194) MSR register. These values may cause macOS boot failure on Intel platforms.

Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required.

5. IgnoreTextInGraphics

Type: plist boolean

Failsafe: false

Description: Select firmwares output text onscreen in both graphics and text mode. This is normally unexpected, because random text may appear over graphical images and cause UI corruption. Setting this option to true will discard all text output when console control is in mode different from Text.

Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required. This option may hide onscreen error messages. ConsoleControl may need to be set to true for this to work.

6. ReplaceTabWithSpace

Type: plist boolean

Failsafe: false

Description: Some firmwares do not print tab characters or even everything that follows them, causing difficulties or inability to use the UEFI Shell builtin text editor to edit property lists and other documents. This option makes the console output spaces instead of tabs.

Note: ConsoleControl may need to be set to true for this to work.

7. ProvideConsoleGop

Type: plist boolean

Failsafe: false

Description: macOS bootloader requires GOP (Graphics Output Protocol) to be present on console handle. This option will install it if missing.

8. ReconnectOnResChange

 $\mathbf{Type}:$ plist boolean

Failsafe: false

Description: Reconnect console controllers after changing screen resolution.

On some firmwares when screen resolution is changed via GOP, it is required to reconnect the controllers, which produce the console protocols (simple text out). Otherwise they will not produce text based on the new resolution.

Note: On several boards this logic may result in black screen when launching OpenCore from Shell and thus it is optional. In versions prior to 0.5.2 this option was mandatory and not configurable. Please do not use this unless required.

$9. \ {\tt ReleaseUsbOwnership}$

 $\mathbf{Type}:$ plist boolean

Failsafe: false

Description: Attempt to detach USB controller ownership from the firmware driver. While most firmwares manage to properly do that, or at least have an option for, select firmwares do not. As a result, operating system may freeze upon boot. Not recommended unless required.

10. RequestBootVarFallback

Type: plist boolean

Failsafe: false

Description: Request fallback of some Boot prefixed variables from OC_VENDOR_VARIABLE_GUID to

 ${\tt EFI_GLOBAL_VARIABLE_GUID}.$

This quirk requires RequestBootVarRouting to be enabled and therefore OC_FIRMWARE_RUNTIME protocol implemented in FwRuntimeServices.efi.

By redirecting Boot prefixed variables to a separate GUID namespace we achieve multiple goals:

- Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
- Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
- Potentially incompatible boot entries, such as macOS entries, are not deleted or anyhow corrupted.

However, some firmwares do their own boot option scanning upon startup by checking file presence on the available disks. Quite often this scanning includes non-standard locations, such as Windows Bootloader paths. Normally it is not an issue, but some firmwares, ASUS firmwares on APTIO V in particular, have bugs. For them scanning is implemented improperly, and firmware preferences may get accidentally corrupted due to BootOrder entry duplication (each option will be added twice) making it impossible to boot without cleaning NVRAM.

To trigger the bug one should have some valid boot options (e.g. OpenCore) and then install Windows with RequestBootVarRouting enabled. As Windows bootloader option will not be created by Windows installer, the firmware will attempt to create it itself, and then corrupt its boot option list.

This quirk forwards all UEFI specification valid boot options, that are not related to macOS, to the firmware into BootF### and BootOrder variables upon write. As the entries are added to the end of BootOrder, this does not break boot priority, but ensures that the firmware does not try to append a new option on its own after Windows installation for instance.

11. RequestBootVarRouting

Type: plist boolean

Failsafe: false

Description: Request redirect of all Boot prefixed variables from EFI_GLOBAL_VARIABLE_GUID to

OC_VENDOR_VARIABLE_GUID.

This quirk requires OC_FIRMWARE_RUNTIME protocol implemented in FwRuntimeServices.efi. The quirk lets default boot entry preservation at times when firmwares delete incompatible boot entries. Simply said, you are required to enable this quirk to be able to reliably use Startup Disk preference pane in a firmware that is not compatible with macOS boot entries by design.

12. SanitiseClearScreen

Type: plist boolean

Failsafe: false

Description: Some firmwares reset screen resolution to a failsafe value (like 1024x768) on the attempts to clear screen contents when large display (e.g. 2K or 4K) is used. This option attempts to apply a workaround.

Note: ConsoleControl may need to be set to true for this to work. On all known affected systems ConsoleMode had to be set to empty string for this to work.

13. UnblockFsConnect

Type: plist boolean

Failsafe: false

Description: Some firmwares block partition handles by opening them in By Driver mode, which results in File System protocols being unable to install.

 $\it Note$: The quirk is mostly relevant for select HP laptops with no drives listed.

12 Troubleshooting

12.1 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- To install Windows, macOS, and OpenCore on the same drive you can specify Windows bootloader path (\EFI\Microsoft\Boot\bootmgfw.efi) in BlessOverride section.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider setting SystemUUID to the original firmware UUID. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license. The nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example ./brigadier.exe -m iMac19,1. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter msiexec /i BootCamp.msi. In case you already have a previous version of Boot Camp installed you will have to remove it first by running msiexec /x BootCamp.msi command. BootCamp.msi file is located in BootCamp/Drivers/Apple directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction FlipFlopWheel must be set to 1 as explained on SuperUser.
- RealTimeIsUniversal must be set to 1 to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known tools are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

Why do I see Basic data partition in Boot Camp Startup Disk control panel?

Boot Camp control panel uses GPT partition table to obtain each boot option name. After installing Windows separately you will have to relabel the partition manually. This can be done with many tools including open-source gdisk utility. Reference example:

```
PS C:\gdisk> .\gdisk64.exe \\.\physicaldrive0
GPT fdisk (gdisk) version 1.0.4

Command (? for help): p
Disk \\.\physicaldrive0: 419430400 sectors, 200.0 GiB
```

Sector size (logical): 512 bytes

Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12

Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	1023999	499.0 MiB	2700	Basic data partition
2	1024000	1226751	99.0 MiB	EF00	EFI system partition
3	1226752	1259519	16.0 MiB	0C01	Microsoft reserved
4	1259520	419428351	199.4 GiB	0700	Basic data partition

Command (? for help): c Partition number (1-4): 4 Enter name: BOOTCAMP

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

```
Do you want to proceed? (Y/N): Y OK; writing new GUID partition table (GPT) to \.\Disk synchronization succeeded! The computer should now use the new partition table. The operation has completed successfully.
```

Listing 3: Relabeling Windows volume

How to choose Windows BOOTCAMP with custom NTFS drivers?

Third-party drivers providing NTFS support, such as NTFS-3G, Paragon NTFS, Tuxera NTFS or Seagate Paragon Driver break certain macOS functionality, including Startup Disk preference pane normally used for operating system selection. While the recommended option remains not to use such drivers as they commonly corrupt the filesystem, and prefer the driver bundled with macOS with optional write support (command or GUI), there still exist vendor-specific workarounds for their products: Tuxera, Paragon, etc.

12.2 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source level debugging with GDB or IDA Pro is also available. For GDB check OcSupport Debug page. For IDA Pro you will need IDA Pro 7.3 or newer, refer to Debugging the XNU Kernel with IDA Pro for more details.

To obtain the log during boot you can make the use of serial port debugging. Serial port debugging is enabled in Target, e.g. 0xB for onscreen with serial. OpenCore uses 115200 baud rate, 8 data bits, no parity, and 1 stop bit. For macOS your best choice are CP2102-based UART devices. Connect motherboard TX to USB UART RX, and motherboard GND to USB UART GND. Use screen utility to get the output, or download GUI software, such as CoolTerm.

Note: On several motherboards (and possibly USB UART dongles) PIN naming may be incorrect. It is very common to have GND swapped with RX, thus you have to connect motherboard "TX" to USB UART GND, and motherboard "GND" to USB UART RX.

Remember to enable COM port in firmware settings, and never use USB cables longer than 1 meter to avoid output corruption. To additionally enable XNU kernel serial output you will need debug=0x8 boot argument.

12.3 Tips and Tricks

1. How to debug boot failure?

Normally it is enough to obtain the actual error message. For this ensure that:

• You have a DEBUG or NOOPT version of OpenCore.

- Logging is enabled (1) and shown onscreen (2): $Misc \rightarrow Debug \rightarrow Target = 3$.
- Logged messages from at least DEBUG_ERROR (0x8000000), DEBUG_WARN (0x00000002), and DEBUG_INFO (0x000000040) levels are visible onscreen: Misc → Debug → DisplayLevel = 0x80000042.
- Critical error messages, like DEBUG_ERROR, stop booting: Misc → Security → HaltLevel = 0x80000000.
- Watch Dog is disabled to prevent automatic reboot: Misc → Debug → DisableWatchDog = true.
- Boot Picker (entry selector) is enabled: Misc → Boot → ShowPicker = true.

If there is no obvious error, check the available hacks in Quirks sections one by one. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell may help to see early debug messages.

2. How to customise boot entries?

OpenCore follows standard Apple Bless model and extracts the entry name from .contentDetails and .disk_label.contentDetails files in the booter directory if present. These files contain an ASCII string with an entry title, which may then be customised by the user.

3. How to choose the default boot entry?

OpenCore uses the primary UEFI boot option to select the default entry. This choice can be altered from UEFI Setup, with the macOS Startup Disk preference, or the Windows Boot Camp Control Panel. Since choosing OpenCore's BOOTx64.EFI as a primary boot option limits this functionality in addition to several firmwares deleting incompatible boot options, potentially including those created by macOS, you are strongly encouraged to use the RequestBootVarRouting quirk, which will preserve your selection made in the operating system within the OpenCore variable space. Note, that RequestBootVarRouting requires a separate driver for functioning.

4. What is the simplest way to install macOS?

Copy online recovery image (*.dmg and *.chunklist files) to com.apple.recovery.boot directory on a FAT32 partition with OpenCore. Load OpenCore Boot Picker and choose the entry, it will have a (dmg) suffix. Custom name may be created by providing .contentDetails file.

To download recovery online you may use macrecovery.py tool from MacInfoPkg.

For offline installation refer to How to create a bootable installer for macOS article.

5. Why do online recovery images (*.dmg) fail to load?

This may be caused by missing HFS+ driver, as all presently known recovery volumes have HFS+ filesystem. Another cause may be buggy firmware allocator, which can be worked around with AvoidHighAlloc UEFI quirk.

6. Can I use this on Apple hardware or virtual machines?

Sure, most relatively modern Mac models including MacPro5,1 and virtual machines are fully supported. Even though there are little to none specific details relevant to Mac hardware, some ongoing instructions can be found in acidanthera/bugtracker#377.

7. Why do Find&Replace patches must equal in length?

For machine code (x86 code) it is not possible to do differently sized replacements due to relative addressing. For ACPI code this is risky, and is technically equivalent to ACPI table replacement, thus not implemented. More detailed explanation can be found on AppleLife.ru.

8. How can I migrate from AptioMemoryFix?

Behaviour similar to that of AptioMemoryFix can be obtained by installing FwRuntimeServices driver and enabling the quirks listed below. Please note, that most of these are not necessary to be enabled. Refer to their individual descriptions in this document for more details.

- ProvideConsoleGop (UEFI quirk)
- AvoidRuntimeDefrag
- DiscardHibernateMap
- EnableSafeModeSlide
- EnableWriteUnprotector
- ForceExitBootServices
- ProtectCsmRegion
- ProvideCustomSlide

- SetupVirtualMapShrinkMemoryMap